Universidade de Aveiro
2009
Departamento de Electrónica e Telecomunicações
e Informática

João Filipe Fernandes Garcia Lima

PROCESSADOR COM CONJUNTO DE INSTRUÇÕES VARIÁVEL REMOTAMENTE

João Filipe Fernandes Garcia Lima

PROCESSADOR COM CONJUNTO DE INSTRUÇÕES VARIÁVEL CONFIGURÁVEL REMOTAMENTE

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Electrónica e Telecomunicações, realizada sob a orientação científica do Prof. Doutor Valeri Skyliarov e do Prof. Doutor Nuno Borges de Carvalho, professores do Departamento de Electrónica e Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais, irmãos e avós.

O júri

Presidente

Prof. Dr. António Manuel de Brito Ferrari Almeida
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Vogais

Prof. Dr. Valeri Skyliarov
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Prof. Dr. Nuno Borges de Carvalho
Professor Associado do Departamento de Electrónica, Telecomunicações e Informática da Universidade do Aveiro

Prof. Dr. Hélio Mendes de Sousa Mendonça
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Agradecimentos

Através deste meio pretendo expressar a minha gratidão para com os meus orientadores, Prof. Doutor Valeri Skyliarov e Prof. Doutor Nuno Borges de Carvalho pela sua tremenda ajuda, apoio e disponibilidade que me prestaram no desenrolar deste trabalho.

Quero expressar igualmente o meu agradecimento aos meus colegas de laboratório Pedro Henriques, Nuno Duarte, Hugo Barreira, Nuno Marujo, Luís Farinha, José Santos, Rui Sancho, António Diogo, Almerindo Paiva e em especial ao meu companheiro Abílio Neves o qual partilhei parte da implementação do projecto disponibilizando sempre ajuda quando necessitei. Gostava também de agradecer ao meu fiel companheiro de grupo e grande amigo Hélder Moura pela sua amizade e ajuda que me ofereceu durante estes 5 anos.

Por último aproveito para agradecer aos meus pais, Felisberto e Maria, aos meus irmãos, Pedro e Duarte, a todos os meus amigos de Aveiro e amigos da minha terra natal, Moimenta da Beira por todo o apoio incondicional que me ofereceram durante o meu percurso académico.

Palavras-chave FPGAs, Processadores, Co-Processadores, SoC, Rádio Frequência.

Resumo

Com as mais recentes evoluções tecnológicas na área de sistemas digitais, traduzidas por um aumento significativo da velocidade e diminuição das dimensões e consumo dos circuitos integrados tem levado à emergência do conceito de sistemas SoC (System On Chip).

Uns dos grandes avanços para este conceito têm sido obtidos graças à evolução significativa dos dispositivos lógicos reprogramáveis, nomeadamente os dispositivos de grande capacidade lógica (VLSI), Destes dispositivos destaca-se a designada FPGA. Esta permite a construção de sistemas digitais complexos, o que devido às suas potentes ferramentas de desenvolvimento permite uma grande flexibilidade bem com um tempo de prototipagem muito reduzido.

A evolução tecnológica tem-se estendido a outras áreas tais como sistemas de rádio. Estes são cada vez mais utilizados em sistemas que se encontram no quotidiano desempenhando funções de controlo remotamente, imagem e vídeo entre muito outros. Com os avanços das FPGAs os sistemas rádio começam igualmente a utilizar estes dispositivos para conseguir por em prática alguns conceitos o que devido às suas características reconfiguráveis é possível ajustar determinado sistema para a recepção de um determinado sinal sem a alteração do hardware.

Visto a crescente evolução dos sistemas digitais reconfiguráveis e sistemas rádio, é de grande interesse a utilização destas duas áreas para construção de sistemas complexos. Por este facto pretende-se com este trabalho integrar as duas componentes.

Neste contexto esta tese tem como principal motivação a proposta de uma nova forma de estruturar a arquitectura de um processador permitindo que o seu conjunto de instruções seja alterado via rádio frequência, poupando recursos na FPGA.

Keywords FPGAs, Processors, Co-Processors, SoC, Radio Frequency.

Abstract

Due to most recently evolutions in digital systems, traduced in an increase of frequency, decrease of dimensions and power, the concept of SoC became a reality. One of the most advances in this new approach of design was obtained due to the recently capabilities of reconfigurable logic devices, including FPGA. It allows the construction of complex digital systems in a short amount of time and with a great flexibility.

The worldwide technologic evolution has affected areas like radio systems presented in most applications today. With advances in FPGA devices and due to their reconfigurable characteristics they are beginning to appear in radio systems due to their ability of changing transmission system without hardware modifications.

Taking into account this evolution there is a great motivation to work with these two technologies. The presented work gives a new example of applicability in integration of reconfigurable and radio systems with a construction of a remote variable instruction set processor.

In this context, the work's main motivation is to propose a new form of structuring processor architecture, allowing remote changes in their instruction set, traducing in reduction of occupied slices in the FPGA.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Enquadramento

As mais recentes evoluções tecnológicas na área de sistemas digitais, traduzidas por um aumento significativo da velocidade e diminuição das dimensões e consumo dos circuitos integrados têm levado à emergência do conceito de sistemas SoC (*System On Chip*). Estes sistemas integram diversos componentes na mesma pastilha de silício, tais como microprocessadores, memórias, conversores analógico-digitais, controladores I/O etc., tornando a construção dos sistemas electrónicos mais baratos, mais eficientes e com menor consumo, o que leva, inevitavelmente, a uma grande utilização por parte dos projectistas.

Os grandes avanços para este conceito têm sido obtidos graças à evolução significativa dos dispositivos lógicos reprogramáveis, nomeadamente os dispositivos de grande capacidade lógica (VLSI), podendo estes alterar a sua função lógica sem a necessidade da troca de circuito integrado, mesmo após o processo de fabrico e, desta forma, serem costumizados para uma determinada aplicação.

Destes dispositivos destaca-se a designada FPGA. Esta possibilita a construção de sistemas digitais complexos, o que devido às suas potentes ferramentas de desenvolvimento permite uma grande flexibilidade, bem como um tempo de prototipagem muito reduzido. Este facto, tem levado à mudança do conceito *general propose* para o conceito *application-domain* ou seja, os sistemas digitais começam a ser construídos para uma aplicação muito específica indo de encontro às necessidades do projectista e não para um conjunto de aplicações (Ex:Microcontrolador PIC), reduzindo desta forma alguma aplicabilidade dos componentes ASIC.

Todas estas evoluções têm levado ao aparecimento, para além de sistemas SoC, sistemas NoC nos quais todos os elementos de um sistema SoC se comportam e comu-

nicam como se estes se encontrassem numa rede de telecomunicações.

A evolução tecnológica tem-se estendido a outras áreas tais como sistemas de rádio. Estes são cada vez mais utilizados em sistemas que se encontram no quotidiano desempenhando funções de controlo remotamente, imagem e vídeo entre muito outros. Com os avanços das FPGAs, os sistemas rádio começam igualmente a utilizar estes dispositivos para conseguir pôr em prática alguns conceitos como o *cognitive radio* uma vez que, devido às suas características reconfiguráveis, é possível ajustar determinado sistema para a recepção de um determinado sinal sem a alteração do *hardware*.

A proposta deste trabalho vem de encontro à utilização destas duas áreas propondo uma nova função para a utilização de FPGAs e rádio frequência.

1.2 Motivação

Com o crescente aumento da capacidade das FPGAs, é possível hoje em dia incorporar nestes processadores com alguma complexidade. Estes podem ser incluídos directamente na pastilha de silício (***Hard-Core***) ou implementados através de um processo de síntese (***Soft-Core***). Devido à grande flexibilidade destes dispositivos é possível acoplar a estes processadores periféricos estando estes no mesmo *chip*. Os periféricos ligados ao processador desenvolvido provêm de componentes existentes em livrarias IP ou são construídos pelo próprio projectista. Concluída a construção do microcontrolador, é possível programá-lo em linguagens comuns como é o caso da C e C++.

No entanto, os processadores possuem, muitas vezes, um *instruction set* de instruções relativamente elevado, quer se trate de processadores embutidos em FPGA ou não. Este conjunto de instruções em grande parte dos programas não é utilizado por completo, o que leva, inevitavelmente, a um desperdício de recursos de uma FPGA, uma vez que, mesmo não sendo aquelas utilizadas, o *hardware* dedicado à sua execução encontra-se implementado. Neste contexto, esta tese tem como principal motivação a proposta de uma nova forma de estruturar a arquitectura de um processador na tentativa de reduzir o tamanho ocupado por um ele numa FPGA, baseando-se para isso na construção de um processador com um *set* de instruções variável, ou seja, apenas as instruções estritamente necessárias à execução de um determinado programa são carregadas para a FPGA.

As instruções não utilizadas deverão, no entanto, estar ao dispor para o caso da sua utilização na execução de novo programa. Este facto, leva-nos à motivação de uma área em constante crescimento e de grande uso no quotidiano: Rádio Frequência. Apesar da grande importância desta nos tempos correntes, os componentes das livra-

rias IP disponibilizadas pelo fabricante Xilinx (FPGAs utilizadas nesta tese) apenas contemplam componentes para comunicações com fios, motivando, desta maneira, a construção de um componente de fácil utilização que faça a interface com um módulo *wireless* específico.

Visto o consumo energético ser uma das características cada vez mais importantes nos dias de hoje é grande importância a escolha de um módulo de baixo consumo.

Com uma interface *wireless* entre FPGAs existe uma grande quantidade de aplicações que se podem desenvolver em torno deste assunto, motivando cada vez mais a interacção remota entre dispositivos. Esta tese propõe igualmente uma nova função para os sistemas rádio, consistindo esta no carregamento de um programa *assembly,* bem como instruções necessárias para a execução do mesmo.

A conjugação destes factos leva a uma melhoria de aproveitamento dos recursos da FPGA, bem como a possibilidade de alteração de um programa sem a necessidade de ferramentas como um computador de uso pessoal e ferramentas de desenvolvimento, aumentando, ainda mais, a flexibilidade já elevada por parte destes dispositivos reconfiguráveis.

A aplicabilidade do tema proposto nesta dissertação pode ser encontrada em basicamente todos os sistemas baseados em FPGA que tenham necessidade do uso de microprocessadores.

1.3 Objectivos

Este trabalho tem como principal objectivo a construção de um processador com um *instruction set* variável, sendo este actualizado via rádio frequência. As etapas da construção deste sistema podem ser decompostas nos seguintes objectivos:

- Compreensão do funcionamento de um processador, especialmente as etapas da execução de instruções;
- Proposta de uma nova abordagem na construção de um processador, nomeadamente da sua unidade de controlo para permitir um *set* de instruções variável;
- Desenvolvimento de um processador com *instruction set* variável, orientado para pesquisa combinatória.
- Interligação do processador desenvolvido com um processador de uso geral;
- Construção de uma interface que permita a comunicação entre FPGAs via rádio frequência;

- Ligação da interface RF ao sistema constituído pelo processador de uso geral e processador construído;
- Envio remoto da informação acerca das instruções para o processador com um *instruction set* variável, bem como alteração do programa em *assembly* do processador de uso geral via rádio frequência;
- Construção de um demonstrador para verificar a funcionalidade do sistema;

1.4 Estrutura da Tese

Esta tese encontra-se dividida em 6 capítulos. Estes encontram-se ordenados de acordo com a ordem de trabalho seguida no desenvolvimento da dissertação. Para além do presente capítulo temos a seguinte sequência:

- **Capítulo 2 - Sistemas digitais reconfiguráveis -** Neste capítulo é apresentado um breve resumo dos sistemas digitais reconfiguráveis. Começa por uma introdução aos vários dispositivos lógicos existentes, na qual é apresentada de uma forma mais extensiva a FPGA, uma vez que esta constituiu o principal componente da realização deste trabalho. Após uma descrição dos componentes principais de uma FPGA são apresentados vários modelos do fabricante Xilinx, as suas principais características e aplicações. Para finalizar, são apresentadas as características únicas da programação HDL.
- **Capítulo 3 - Sistemas Computacionais e Ferramentas de Apoio -** Neste capítulo são apresentados os conceitos fundamentais a ter em conta na construção de um processador e ferramentas de apoio utilizadas. Este começa por uma introdução aos diversos sistemas computacionais. De seguida, é apresentada a organização básica de um computador, a qual incorpora o processador. De forma a compreender o funcionamento deste, é apresentado um exemplo concreto (MIPS), o que permitiu obter bases essenciais sobre o funcionamento de um processador. De seguida, são apresentados os motivos principais para a utilização de FPGAs para incorporação de microprocessadores, bem como exemplo de processadores já desenvolvidos para estas. Para finalizar, é apresentado algum trabalho relacionado bem como as ferramentas de apoio utilizadas.
- **Capítulo 4 - Processador com um *instruction set* variável -** Neste capítulo é apresentado o processador desenvolvido. Este, começa pela abordagem e raciocínio adoptado na construção deste processador. De forma a conseguir obter

uma unidade de controlo variável, é apresentada uma máquina de estados finitos reprogramável, a qual desempenha uma função decisiva na construção do componente proposto. De seguida, é apresentado o processador construído, bem como a explicação de todo o seu processo de actualização das instruções do mesmo. Finalmente, é descrito também o processador de uso geral ao qual foi ligado o processador combinatório desenvolvido e apresentam-se alguns resultados sobre o sistema.

- **Capítulo 5 - Interacção remota -** Neste capítulo é apresentada toda a interface desenvolvida para comunicação sem fios. Começa com a apresentação do módulo *wireless* e das suas características mais relevantes. De seguida, são explicados todos os conceitos fundamentais para a construção de uma interface com o módulo, assim como todos os componentes desenvolvidos. Para finalizar, é apresentada a ligação da interface construída com o restante sistema computacional e ainda a explicação do procedimento para a envio de novas instruções e programa *assembly* remotamente bem como resultados práticos obtidos.

- **Capítulo 6 - Conclusão e Trabalho Futuro -** Este capítulo completa a dissertação. É apresentado um breve resumo do trabalho e principais conclusões. Para terminar são apresentadas algumas propostas para trabalho futuro.

- **Anexo A - Instruções do Co-Processador e Processador de uso geral -** Este apêndice descreve as diversas instruções do Co-Processador e Processador de uso geral. Inicia-se com uma explicação detalhada do fluxo da unidade de controlo para a execução de cada uma das instruções desenvolvidas. Para finalizar, são apresentadas as instruções que compõem o processador de uso geral utilizado.

- **Anexo B - CC1101 -** Este apêndice descreve informação adicional sobre o *transceiver* utilizado para a realização deste trabalho.

Capítulo 2

Sistemas digitais reconfiguráveis

2.1 Sumário

Devido à grande evolução dos dispositivos lógicos programáveis, designados frequentemente por PLD (*Programable Logic Devices*), é possível, hoje em dia, projectar sistemas digitais de grande complexidade, recorrendo a ferramentas que se encontram ao alcance de um cidadão comum. Neste capítulo, serão abordados alguns dos principais dispositivos lógicos programáveis e a sua utilização nos dias de hoje. Uma vez que o principal componente lógico utilizado neste projecto foi uma FPGA, será feita neste capítulo uma maior incidência sobre este componente, sendo apresentados os variados tipos de FPGAs existentes, as suas aplicações e os principais fabricantes.

2.2 Introdução

Um dispositivo lógico programável (PLD) é um dispositivo electrónico que é utilizado para a construção de sistemas digitais reconfiguráveis. Estes dispositivos permitem que uma determinada função lógica desejada, seja programada depois da sua saída de fábrica. Isto permite uma maior flexibilidade no projecto de sistemas digitais. Estes dispositivos encontram-se divididos nas seguintes categorias:

- **SSI** *(Small Scale Integration)* – Contêm até cerca de uma dezena de portas lógicas independentes em que as entradas e saídas se encontram disponibilizadas para o exterior.[29]

- **MSI** *(Medium Scale Integration)* – Contêm entre 10 e 100 portas lógicas, executando uma determinada função elementar.[29]

- **LSI** *(Large Scale Integration)* – Possuem entre 100 e alguns milhares de portas lógicas. Estes componentes, já conseguem desempenhar uma função lógica com pouca complexidade.[29]
- **VLSI** *(Very Large Scale Integration)* – Dispositivos em que o número de portas lógicas é da ordem dos milhões. As FPGAs constituem um exemplo de estes tipos de dispositivos.[29]

A escolha da categoria para um determinado projecto depende directamente da aplicação a que se destina. Dentro dos dispositivos lógicos programáveis, são ainda possível diferencia-los segundo as seguintes categorias:

- *Mask-Programmable e field-programmable.*
- Capacidade de o seu conteúdo ser apagado ou não (*erasable* ou *non-erasable*).

Dispositivos do tipo *Mask-programmable* apenas podem ser programados pelo fabricante na sua construção. Este tipo de dispositivo possuem menos atraso que os dispositivos *field-programmable,* uma vez que, as ligações não podem ser alteradas no final da primeira programação. Estes dispositivos são caros. No entanto, quando em grande quantidade, possuem um custo muito baixo.

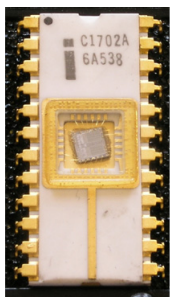

Figura 2.1: EPROM Intel 1702 do tipo *Mask - Programmable*[1]

Já os dispositivos *Field-Programmable* podem ser programados pelo cliente final. Estes *chips* são mais baratos e podem ser programados em qualquer momento. No entanto, quando em grande quantidade, acabam por possuir um custo mais elevado que os dispositivos *Mask-Programmable.*

Figura 2.2: FPGA do tipo *Field Programmable*[2]

2.3 Tipos de dispositivos lógicos programáveis

Os dispositivos lógicos programáveis podem ser divididos em três principais classes: **SPLD** (*Simple PLD*), **CPLD** (*Complex PLD*) e **FPGA** (*Field Programmable Gate Array*). Dentro do tipo **SPLD** podemos ainda classificar os seguintes tipos de dispositivos lógicos: **PLA** (*Programmable Logic Array*), **PAL** (*Programmable Array Logic*), **GAL** (*Generic Logic Array*) e **PROM** (*Programmable ROM*).

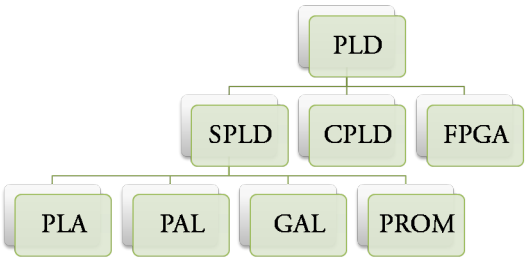

Figura 2.3: Tipos de dispositivos lógicos programáveis

2.3.1 Dispositivos SPLD

Os dispositivos SPLD constituem os dispositivos lógicos programáveis mais simples e mais baratos. Estes componentes são usados em aplicações comerciais, industriais e de comunicações. Possuem, usualmente, cerca de 4 a 22 macro-células, sendo estas constituídas por portas lógicas AND e OR e um Flip-Flop, conseguindo-se, deste modo, criar uma função booleana pequena em cada macro-célula. Assim, torna-se possível através de uma dada entrada e de acordo com uma função lógica, formar uma determinada saída, podendo esta ser guardada no flip-flop até à próxima transição do relógio.

PROM - Programmable ROM

As PROM constituem uma forma básica de guardar informação digital, podendo utilizar-se para desempenhar uma determinada função lógica, considerando-se, deste modo, como um SPLD. Cada bit neste componente encontra-se trancado por um fusível ou anti-fusível. Este dispositivo apenas pode ser programado uma vez, sendo neste procedimento aplicada uma voltagem elevada, levando a um arrebentamento do fusível ou anti-fusível, conseguindo-se, desta forma, obter um componente com valores lógicos permanentes. Este componente pode ser do tipo *Mask-Programmable* ou do tipo *Field-Programmable*. Existem igualmente memórias PROM que podem ser reescritas, designando-se neste caso por **EPROM** (*Erasable Programmable Read Only Memory*) ou por **EEPROM** (*Electrically Erasable Programmable Read Only*). A diferença entre

os dois tipos apresentados, consiste na forma como as memórias são apagadas. No primeiro caso, este processo é feito através da exposição à luz ultra-violeta enquanto no segundo, este processo é feito electricamente. [30]

PLA - Programmable Logic Array

Uma PLA é um dos tipos de dispositivos da categoria SPLD. Esta permite a implementação de circuitos combinatórios. É constituída por um plano de portas AND e outro de portas OR. A ligação dos dois planos permite a implementação de uma função lógica. A saída consiste, deste modo, numa soma de produtos. A estrutura deste dispositivo encontra-se representada na figura seguinte.[30]

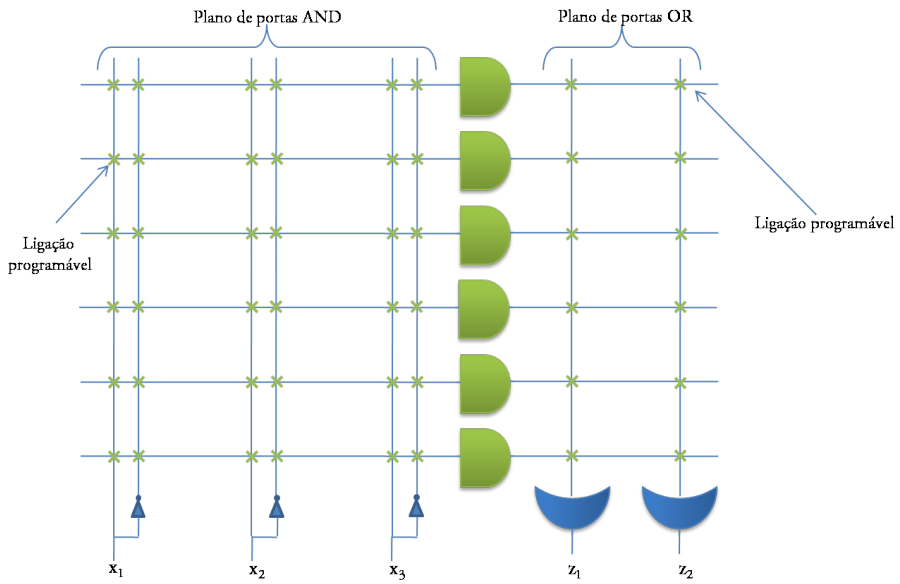

Figura 2.4: Estrutura de uma PLA

PAL – Programmable Array Logic

Ao contrário da PLA que possui dois planos sendo ambos programáveis, a PAL possui apenas o plano de portas lógicas AND programável, sendo o plano OR fixo. Assim, a saída é um conjunto de soma de produtos em que estes são programáveis e as somas são fixas. Este circuito tem a desvantagem de possuir uma menor flexibilidade em relação ao dispositivo lógico PLA. A sua estrutura encontra-se representada na figura seguinte.[30]

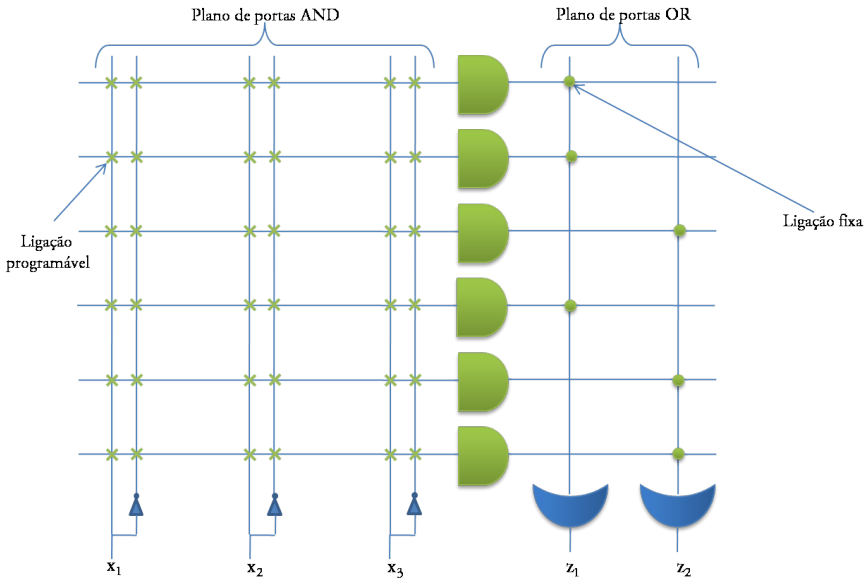

Figura 2.5: Estrutura de uma PAL

GAL – Generic Logic Array

Este consiste numa evolução do dispositivo PAL. A sua estrutura é idêntica, podendo, no entanto, ser reprogramável. Esta é a principal vantagem deste dispositivo.

Figura 2.6: Lattice GAL [3]

2.3.2 CPLD (*Complex Programmable Logic Device*)

Os CPLD aparecem para satisfazer a necessidade de uma maior quantidade de blocos lógicos. Estes pertencem à categoria LSI, possuindo, deste modo, uma densidade de portas lógicas relativamente elevada. A sua estrutura é apresentada na figura seguinte.

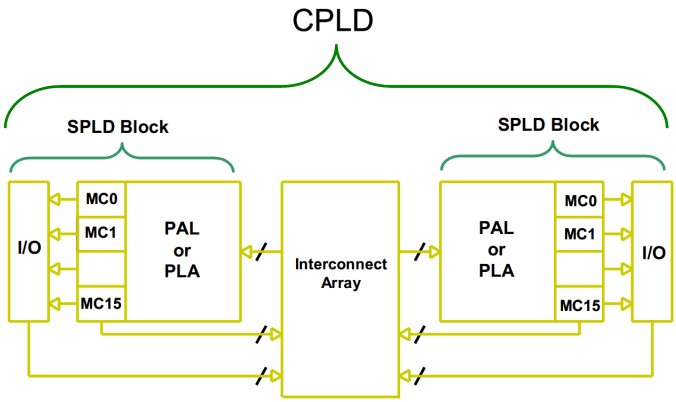

Figura 2.7: Estrutura do CPLD [4]

O conceito principal dos CPLDs consiste no agrupamento de vários blocos do tipo SPLD, podendo estes ser do tipo PAL ou PLA, no mesmo chip. Conforme a complexidade da função lógica que deverá ser desempenhada, serão utilizados apenas os blocos separados ou então será feita uma ligação entre blocos, conseguindo-se, desta forma, a elaboração de uma função com uma grande complexidade. Esta interligação é feita através do bloco ***Interconnect Array***. Para além deste bloco, existem ainda blocos I/O para cada um dos blocos SPLD. Destacam-se então as principais características do CPLD: [4]

- Uma unidade central para interligação de blocos SPLD (*Interconnect Array*);
- *Routing* simples, isto é, as ligações internas são simples;
- Rápido;

Aplicabilidade dos dispositivos CPLD

Os CPLDs possuem uma grande facilidade a nível de design, custos de desenvolvimento baratos, conseguindo-se desta maneira aplicar estes dispositivos no mercado de uma forma rápida e eficiente. Estes dispositivos são utilizados, actualmente, na área das comunicações sem fios, comunicações industriais, comunicações móveis e computação gráfica, entre outros. [31]

2.3.3 FPGA (*Field Programmable Gate Array*)

Uma FPGA consiste num dispositivo semicondutor programável baseado em matrizes de blocos lógicos configuráveis (CLBs) interligados via ligações programáveis. A partir da interligação destes blocos é possível criar uma enorme rede de blocos lógicos,

obtendo-se um circuito de grande densidade e com grande flexibilidade. Ao contrário dos ASIC (*Application Specific Integrated Circuits*), que são dispositivos construídos de fábrica para uma aplicação específica, as FPGAs permitem, devido aos seus componentes reconfiguráveis, construir sistemas depois do seu fabrico e serem alterados quando o utilizador bem entender. Oferece, deste modo, uma maior portabilidade em relação aos ASICs.[4]

2.3.3.1 Evolução

Em meados dos anos 80 a maioria dos sistemas digitais eram implementados integrando diversos componentes *standard*, tais como: microprocessadores, controladores I/O, etc.. Este facto, levava à construção de um sistema com grandes dimensões, uma vez que para um sistema de grande complexidade era necessária uma grande quantidade de componentes, levando à inevitável diminuição da frequência máxima de trabalho. Apareceram, então, o que se designou por circuitos integrados costumizados. Estes eram construídos para incorporar os diversos componentes necessários num único *chip*. Devido à sua costumização este processo era demoroso e caro, tornando-o pouco atraente para pequenas quantidades.

Para combater estes problemas apareceram as designadas FPGAs, introduzidas pelo fabricante Xilinx em 1985 [32]. Estas permitem o desenvolvimento de sistemas digitais de grande complexidade num único *chip* sendo o tempo de projecto muito curto devido às potentes ferramentas CAD.

A figura seguinte apresenta a evolução das arquitecturas das FPGAs.

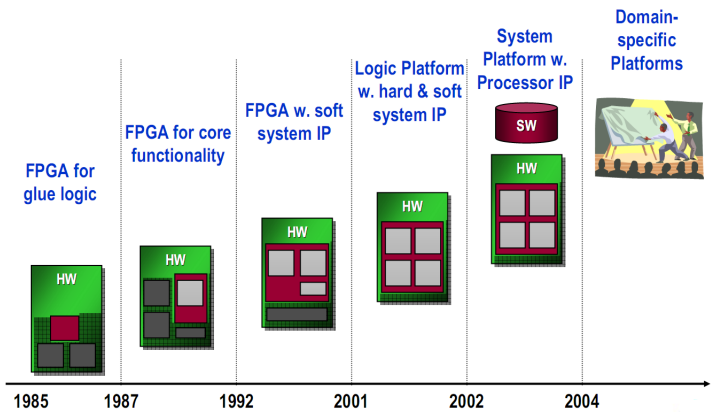

Figura 2.8: Evoluçao da arquitectura das FPGAs [5]

Observando a figura anterior, é possível verificar a grande evolução nas arquitecturas da FPGAs sendo, actualmente, muito utilizada para a implementação de processadores através de livrarias IP podendo ser acoplados a estes periféricos, encontrando-se todos

os componentes dentro do mesmo *chip* e programados posteriormente através de linguagens comuns como C e C++, dando uma grande flexibilidade no desenvolvimento de aplicações.[33]

2.3.3.2 Estrutura interna de uma FPGA

Analisemos, de seguida, analisar com mais pormenor a estrutura interna de uma FPGA, uma vez que esta irá ser um dos pontos centrais deste trabalho.

A estrutura genérica deste componente encontra-se representada na figura seguinte:

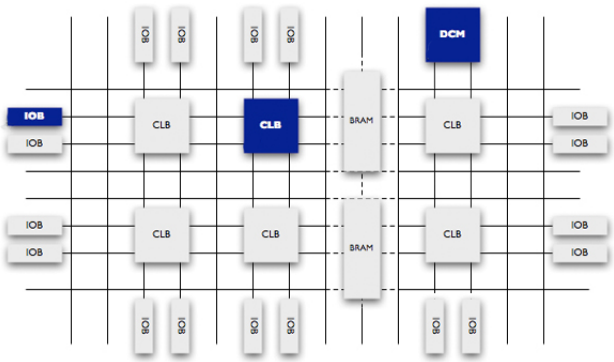

Figura 2.9: Estrutura interna de uma FPGA[6]

CLB (*configurable logic blocks*)

Este componente constitui a unidade principal de uma FPGA. O seu número depende muito de dispositivo para dispositivo. Este consiste numa matriz configurável com 4 ou 6 entradas, alguns circuitos de selecção e flip-flops. A figura 2.10 apresenta um esquema dos componentes que se encontram no CLB.

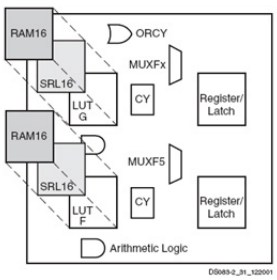

Figura 2.10: Bloco CLB (*configurable logic blocks*) [6]

Esta matriz é reconfigurável, o que implica uma grande flexibilidade e pode então ser configurada para efectuar uma função de RAM, lógica combinatória ou de *shift - register*.[34]

IOB (*Input/Output Block*)

Actualmente as FPGAs possuem dezenas de portas de entrada/saída, conseguindo-se, desta forma, uma forte interacção com outros sistemas. Os blocos de I/O encontram-se agrupados por blocos independentes entre si. A figura seguinte demonstra este facto.

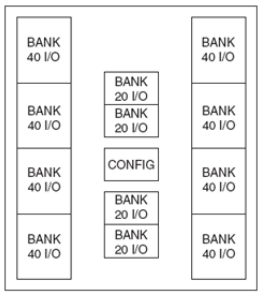

Figura 2.11: IOB (Input/Output Block) [6]

Observando a figura anterior, verifica-se igualmente a existência de uma unidade de configuração. Esta irá permitir uma programação dos blocos I/O, ou seja, programar os pinos I/O como sendo uma entrada, saída ou entrada e saída.[34]

DCM (*Digital Clock Manager*)

Nas primeiras versões das FPGAs havia um grande problema inerente ao relógio. O *designer* na implementação de qualquer sistema tinha de ter especial atenção a problemas de relógio como por exemplo o *skew*. Este fenómeno leva a que o sinal de relógio não chegue a todos os componentes síncronos ao mesmo tempo. É um problema que tem como causa o tamanho da ligação entre dois pontos, o que para frequências elevadas se torna mais significativo. Existem igualmente outros fenómenos que aumentam a gravidade deste problema tais como variações de temperatura e acoplamentos capacitivos quer nas linhas, quer nos componentes. O DCM, adicionado recentemente nas FPGAs permite uma diminuição destes efeitos, reduzindo desta forma uma das grandes preocupações dos projectistas de sistemas reconfiguráveis.[34]

Ligações Programáveis

O bloco CLB permite efectuar uma operação lógica. No entanto, para formar uma rede lógica de grande densidade é necessária a utilização de vários blocos CLB interligados, bem como a sua ligação com blocos de I/O. Estas ligações encontram-se em torno de todos os blocos CLB tal como mostra a figura seguinte. Estas ligações são programáveis,

sendo a programação destas feita automaticamente pelas ferramentas CAD (Ex: *Xilinx ISE*), facilitando o trabalho realizado pelo *designer.*

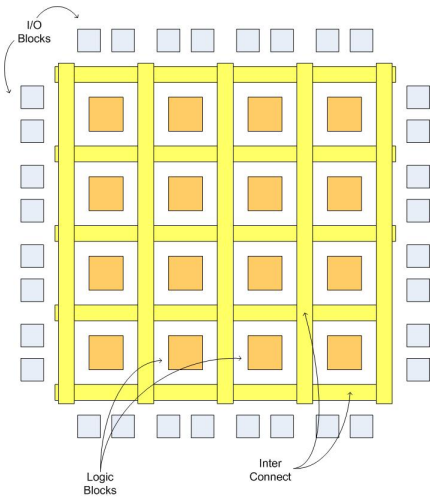

Figura 2.12: Ligaçoes programáveis em FPGA [7]

O facto de haver uma grande quantidade de ligações possíveis para cada CLB permite que efeitos indesejáveis (ex. *Clock Skew)* nos sinais globais, tais como o *clock* sejam reduzidos. [34]

Memória

A maioria das FPGAs actuais possui incorporadas memórias RAM, designadas mais frequentemente por *Block RAM*. A sua quantidade depende de dispositivo para dispositivo. Estas permitem uma diminuição dos blocos CLB utilizados, facilitando deste modo o *design* de sistemas digitais.[34]

2.3.3.3 Aplicações

Actualmente o domínio das FPGAs encontra-se em grande evolução. Devido à sua versatilidade cada vez mais é adoptada, conseguindo-se, com um custo relativamente reduzido, construir plataformas e sistemas que podem ser reconfiguráveis após a saída de fábrica. A aplicabilidade das FPGAs encontra-se nas seguintes áreas:

- **Aeronáutica/Militar -** são utilizadas para efectuar processamento digital de imagem bem como para utilização em aplicações *Software Defined Radio.* [35]

Figura 2.13: Controlador TTP implementado em FPGA [8]

- **Automóvel** - Implementa sistemas de ajuda ao condutor, sistemas de conforto e sistemas de comunicação.[35] Ex: Controlador TTCAN desenvolvido pela Bosch. [36]

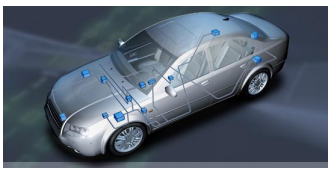

Figura 2.14: Controlador TTCAN implementado em FPGA [9]

- **Broadcast** - Implementa funções que auxiliam na transmissão especialmente de áudio e vídeo.[35]

- **Consumidor** - Começam a desempenhar funções a nível doméstico, tais como controlo de uma rede interna de uma habitação, dispositivos electrónicos etc.[35]

- **Industrial, Científica e Médica** - Desempenham funções na automação industrial, controlo de motores e processamento de imagem em dispositivos médicos.[35]

- **Comunicações sem fios** - Desempenham funções a nível de processamento em banda base, RF, controlo de fluxo de informação bem como implementação de camadas protocolares tais como WCDMA, HSPDA, WiMAX etc.[35]

- **Comunicações com fios** - As funções são em tudo semelhantes à anterior. No entanto, neste caso o ambiente é com fios.[35]

Figura 2.15: Placa de Desenvolvimento Digilent NetFPGA optimizada para redes Ethernet [10]

2.3.3.4 Principais fabricantes e Produtos

O mercado da fabricação de FPGAs encontra-se neste momento em desenvolvimento. Hoje em dia são ainda poucas as empresas a fabricar este tipo de dispositivos. O mercado encontra-se repartido essencialmente por 5 fabricantes: Xilinx, Altera, Lattice Semiconductor, Actel e QuickLogic.

Apareceram, recentemente, no mercado os fabricantes Cypress, Achronix e SiliconBlue. Tendo sido a Xilinx o grande impulsionador das FPGAs, este é o maior fabricante de FPGAs seguido da Altera. Visto ter sido usada uma FPGA da Xilinx abordarei, de seguida, os principais produtos deste fabricante. [37]

Produtos Xilinx

Os produtos disponíveis pelo fabricante Xilinx possibilitam a aplicação de FPGAs em praticamente todas as áreas. Sendo a sua gama de produtos muito elevada, vamos de seguida apenas apresentar os produtos que o próprio fabricante considera de destaque.

Nova Geração de FPGAs - Virtex 6 e Spartan 6

Recentemente, o fabricante Xilinx lançou o que considera a nova geração de FPGAs. Estas sobressaem em relação às suas antecessoras, visto o seu processo de fabrico ter passado para 40nm (Virtex 6) e 45nm (Spartan 6). Este facto leva a uma menor área de silício ocupada, tornando estes componentes, segundo o fabricante, cerca de 50% mais baratos que os antecessores. [28]

Seguidamente, são apresentadas as características mais relevantes de cada um dos *chips*:

Modelo	Características
Spartan 6	• Processo de fabrico de 45 nm • Permite hibernação • Taxa de transferências até 1050 Mb/s data para pinos I/O diferenciais • Drive de corrente programavél até 24mA por pino, • Interfaces serie de alta velocidade:Serial ATA, Aurora, 1G Ethernet, PCI Express, OBSAI, CPRI, EPON, GPON, DisplayPort, e XAUI • Suporte para memórias DDR, DDR2, DDR3, e LPDDR • *Look Up Tables* com 6 entradas. • Block RAMs com 18 Kb cada podendo cada uma delas ser programada como duas Block RAMs de 9Kb independentes
Virtex 6	• Processo de fabrico de 40 nm • *Look Up Tables* com 6 entradas. • Memória distribuida até 64 bits • Block RAMs com 36Kb cada • Pinos I/O trabalham com 1.2V até 2.5V • Bloco Ethernet incorporado para velocidades de 10/100/1000 Mb/s • Blocos Integrados para PCI Express

Tabela 2.1: Principais caracteristicas das FPGAs Spartan 6 e Virtex 6[28]

Virtex 5

A Virtex 5 foi pioneiro a nível de FPGAs de 65nm sendo igualmente a primeira a suportar tensões de alimentação de 1V, possuindo desta forma um consumo energético reduzido.

Figura 2.16: Virtex 5 LX [11]

Possui cerca de 330.000 células lógicas, 1.200 pinos de entrada e saída, 48 transceivers de baixo consumo (RocketIO™ - Transceiver desenvolvido pela Xilinx capaz de taxas de transmissões 6.5Gbps), capacidade de incorporar um processador PowerPC440, suporta PCI Express e possui blocos destinados a comunicação Ethernet. Estas características dependem do sub-tipo desta família.[38]

A família Virtex 5 divide-se em 5 subtipos:

- **Virtex 5 LX** – É concebida para o desenvolvimento em que seja necessária uma grande performance para execução de operações lógicas.[38]
- **Virtex 5 LXT** – Possui igualmente uma grande performance a nivél lógico, bem como um baixo consumo em comunicações série (Transceivers Rocket IO).[38]

- **Virtex 5 SXT** – Possui, também, um baixo consumo em comunicações serie. É igualmente optimizado para efectuar processamento digital de sinal, bem como aplicações que necessitem de uma grande quantidade de memória.[38]
- **Virtex 5 FXT** – Desenvolvido para aplicações que necessitem de uma taxa de transmissão elevada e para sistemas embutidos.[38]
- **Virtex 5 TXT** – Para aplicações que necessitem de uma grande largura de banda, tal como *bridging*, *switching*, aplicações em sistemas de telecomunicações e transferência de dados.[38]

Spartan 3A

A gama Spartan constitui uma solução mais económica, quando a exigência das aplicações não é muito elevada. Esta gama pode possuir entre 50.000 e 3.4 milhões de gates lógicas.[39]

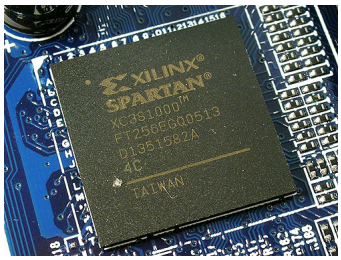

Figura 2.17: Spartan 3A [12]

Tal como na Virtex, dentro da gama Spartan existem algumas gamas e subgamas, tal como Spartan 3E. No entanto, o produto de destaque actualmente é a Spartan 3A. Dentro desta gama existem três modelos:

- **Spartan 3A** – Esta FPGA é considerada a mais barata, possuindo optimizações para aplicações que exijam uma grande quantidade de recursos I/O. Pode ter entre 50K a 1.4 Milhões de gates lógicas e entradas I/O desde 108 a 502 entradas/saídas. As aplicações actuais deste dispositivo encontram-se principalmente na área de sistemas embutidos, sendo capaz de incorporar o processador MicroBlaze da Xilinx.[39]
- **Spartan 3AN** – Esta FPGA possui também uma grande quantidade de portas lógicas, com números iguais ao do modelo anterior. No entanto, esta não se destina a aplicações de grande exigência de I/O, mas sim às não voláteis, ou seja, aplicações em que uma determinada informação não se perca quando

do encerramento do sistema. Para tal, este dispositivo possui uma memória *flash* incorporada com 11Mb, sendo a maior memória *on-chip* actualmente em FPGAs.[39]

- **Spartan 3A DSP** – Tal como indica o nome, este modelo tem como finalidade aplicações que envolvam uma grande quantidade de processamento digital de sinal. Possui 40BRAM (*Block Ram*) por bloco lógico, sendo ideal para os algoritmos de processamento de sinal e de coprocessamento. Este dispositivo consegue, igualmente, incorporar o processador MicroBlaze da Xilinx. [39]

2.4 Linguagens de descrição de Hardware (HDL)

As linguagens de descrição de hardware encontram-se numa classe especial das linguagens de programação. Estas, ao contrário das mais comuns linguagens de programação (Ex: Java), têm como objectivo fazer uma descrição de um circuito electrónico. A sua principal diferença encontra-se na sua sintaxe, permitindo trabalhar directamente com múltiplos processos (paralelismo) e com expressões temporais (Ex: Transição ascendente do relógio). Estas linguagens servem para simulação de circuitos digitais, podendo a descrição ser, mais tarde, sintetizada e implementada em *hardware* (Ex: FPGAs).[40] As principais linguagens utilizadas são VHDL e Verilog. No entanto, hoje em dia é possível já programar em linguagens de mais alto nível como Matlab e C++, sendo estas posteriormente convertidas para uma linguagem de descrição de hardware.[41]

Capítulo 3

Sistemas Computacionais e Ferramentas de Apoio

3.1 Sumário

O trabalho realizado encontra-se, em grande medida, relacionado com arquitectura de computadores. Neste capítulo são apresentados os conceitos fundamentais sobre os sistemas computacionais. Este começa com uma breve introdução aos diversos sistemas existentes. De seguida, são apresentados os conceitos básicos sobre o *instruction set* de um processador e as diferentes fases que este toma para a execução das instruções. Será considerada a arquitectura do processador MIPS como exemplo para uma melhor percepção dos conceitos. Serão igualmente apresentados alguns exemplos de microprocessadores implementados em FPGA assim como algum trabalho relacionado. A terminar, serão apresentadas todas as ferramentas de apoio a este trabalho fazendo, deste modo, uma ponte entre a parte teórica e prática desta dissertação.

3.2 Introdução

Devido aos avanços na área da microelectrónica, os sistemas computacionais fazem hoje parte da nossa sociedade. Estes têm sofrido uma grande explosão a nível de capacidade e potência, tornando-os, actualmente, indispensáveis. Esta evolução tem seguido a conhecida **Lei de Moore**, que nos diz que o número de transístores que podem ser colocados num circuito integrado aumenta exponencialmente, ou seja, duplica aproximadamente a cada dois anos.

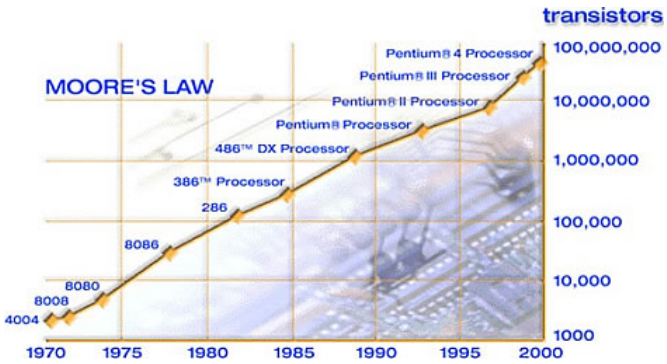

Figura 3.1: Lei de Moore [13]

É possível, actualmente, dividir os sistemas computacionais em três grandes tipos: de uso geral, servidores e embutidos.

- **Sistemas de uso geral** – São os sistemas computacionais mais conhecidos. O exemplo mais evidente é o computador pessoal caracterizado pela sua boa performance relativamente ao seu baixo custo. São considerados os grandes impulsionadores na área de sistemas computacionais.[14]

- **Servidores** – São orientados para grandes cargas de informação. Estas provêm de multiutilizadores que se encontram tipicamente numa rede e que fazem pedidos de processamento simultaneamente. Estes sistemas computacionais utilizam a mesma tecnologia que os sistemas de uso geral, contendo, no entanto, uma grande capacidade de *Input/Output*. Os servidores possuem dois extremos a nível de capacidade e custo. Estes podem este ser um simples computador de uso geral que serve para armazenar dados, aplicações pequenas ou uma página de internet pequena. No outro extremo, encontram-se os super-computadores possuindo uma grande performance e um custo muito elevado. Estes possuem de centenas até milhares de processadores e possuem uma capacidade de memória que pode chegar até aos petabytes. Usualmente, são ligados como servidor e utilizados para cálculos científicos de grande complexidade, tais como previsões meteorológicas.[14]

- **Sistemas Embutidos** – São hoje o maior grupo de sistemas computacionais e utilizados numa enorme quantidade de aplicações. Consiste num sistema computacional desenhado para desempenhar uma ou poucas tarefas, isto é, é um sistema construído para satisfazer necessidades específicas podendo incluir Hardware e componentes mecânicos. Uma vez que estes sistemas são especializados para uma determinada tarefa, os projectistas conseguem facilmente optimizar o

sistema reduzindo o tamanho e o custo deste. Exemplos de sistemas embutidos vão desde PDAs, leitores MP3, máquinas de lavar, aparelhos médicos etc. Deste modo, um sistema embutido possui sempre uma grande quantidade de periféricos, permitindo que a função desejada seja construída e controlada usualmente por um microprocessador. Ao conjunto do microprocessador mais periféricos designa-se por microcontrolador. Se for utilizada uma rede de sistemas embutidos estes designam-se por sistemas distribuídos.[14]

Em qualquer sistema de computação, apenas é possível executar instruções muito simples, pelo que se torna necessária a construção de camadas de software que permitam obter uma abstracção para o programador. Podemos então dividir as seguintes camadas: ***Hardware***, **Sistemas de *Software*** e **Aplicações de *Software***. Estas encontram-se hierarquizadas, tal como mostra a figura seguinte.

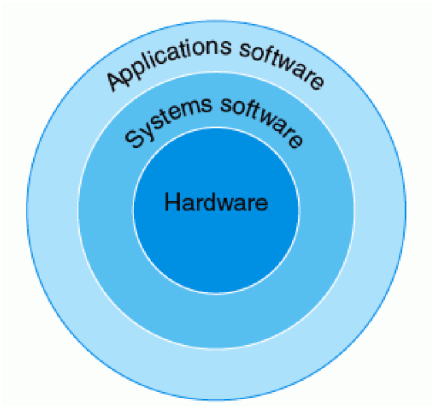

Figura 3.2: Tipicas camadas de um sistema computacional[14]

A camada de mais baixo nível (***Hardware***) permite a implementação de funções simples, fornecendo, desta forma, a estrutura base de um sistema computacional. A camada seguinte tem como objectivo tornar a construção de aplicações computacionais mais simples, fornecendo ferramentas úteis para a construção de uma aplicação e tornando a camada de *hardware* abstracta para o programador. Uma das ferramentas mais importantes é o que designamos por **sistema operativo,** que permite gerir todos os recursos disponíveis de uma forma abstracta para o programador. Incluem-se igualmente nesta camada os **compiladores,** que permitem traduzir um programa numa linguagem de mais alto nível (maior abstracção) numa linguagem que o hardware consegue executar (***assembly***). Este código assembly é traduzido, posteriormente, para linguagem binária, ou seja, a linguagem da máquina, para que possa ser executado. Esta tarefa é feita pelo que designamos por ***assemblers***. Por fim, possuímos a camada

applications sofware o qual incorpora todo o *software* que é desenvolvido por um programador e em que o grau de abstracção é muito elevado. [14]

3.3 Organização de um computador

Um computador digital possui como função primária o processamento de informação vinda de entradas e produção de resultados/saídas consoante um determinado algoritmo. Independentemente do tipo de estrutura que um determinado computador possui, existem alguns componentes que são comuns a todos os sistemas computacionais, desempenhando todos as mesmas funções básicas. Estes componentes encontram-se ilustrados na figura seguinte.

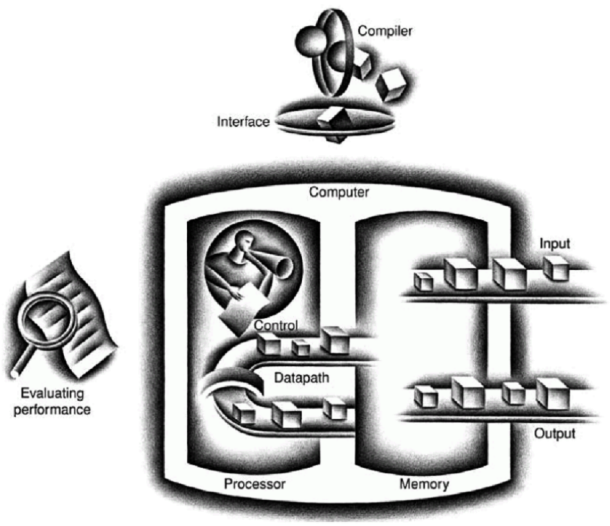

Figura 3.3: Organização de um computador [14]

Observando a figura anterior, é possível verificar 5 grandes componentes de um sistema computacional: unidade de controlo, *datapath*, memória, input e output. Os componentes ***input/output***, tal como os nomes indicam, permitem receber ou transmitir informação para o computador. Dispositivos como um rato, teclado ou scanner são exemplo de componentes *Input*. Já o ecrã ou colunas de som são exemplo de componentes output. Existem igualmente dispositivos *input/output,* como é o caso de *flash drives*, gravadores de cd's, etc. A memória, por sua vez, permite armazenar o programa a ser executado, bem como armazenar resultados intermédios importantes. Esta informação é, usualmente, guardada no que se designa por memória **DRAM**. Dentro do processador, existe outro tipo de memória designada por **memória *cache***. Esta, possui um tamanho muito reduzido relativamente à memória DRAM mas uma grande

velocidade. Tem como função reduzir o tempo de acesso a informação que se encontra na DRAM. Esta, contém a informação mais recentemente utilizada, pelo que o processador não necessita de aceder à memória DRAM se esta se encontrar na memória *cache*. Esta memória é do tipo **SRAM**, as quais são mais caras e ocupam mais espaço do que as memoria DRAM pelo que a capacidade utilizada normalmente anda à volta dos MBytes. Estas, possuem a vantagem de serem muito mais rápidas que as memórias DRAM. Finalmente, existe o processador ou **CPU** (*Central Processor Unit*), o qual é responsável pela execução do programa que se encontra em memória. Este pode ser dividido em dois componentes: *datapath* e unidade de controlo. O *datapath* permite efectuar operações aritméticas, enquanto a unidade de controlo efectua o controlo de sinais do *datapath*, memória e I/O em conformidade com as instruções do programa a executar. [14]

3.3.1 *Instruction Set Architecture (ISA)*

Para que seja possível a construção de programas, existe a necessidade de uma interface entre o hardware e programador. Esta interface designa-se por ***Instruction Set Architecture*** (ISA) e define as palavras que a máquina entende, sendo o seu vocabulário designado por ***Instruction Set*** do processador. Define, igualmente, os modos de endereçamento possíveis, os tipos de dados, os registos (de uso geral e reservados) e memória reservados para o programador. [42]

Tal como foi dito, o programa a ser executado encontra-se em memória. Para além desta informação, dados necessários à execução do programa (Ex: *Arrays* de inteiros) são igualmente armazenados em memória. Podem-se distinguir dois tipos de organização do sistema computacional no que diz respeito à organização da memória: **Arquitectura de Von Neumann** e **Arquitectura de Harvard**.

No primeiro tipo de arquitectura, a memória é utilizada simultaneamente para armazenar o programa e dados. Esta encontra-se representada na figura3.4

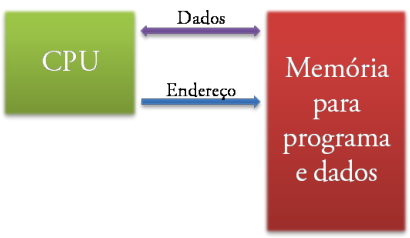

Figura 3.4: Arquitectura de Von Neumann

No segundo tipo de arquitectura, o programa e os dados são armazenados em me-

mórias diferentes. Esta encontra-se representada na figura 3.5.

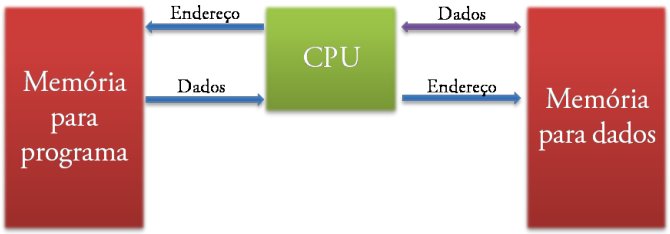

Figura 3.5: Arquitectura de Harvard

Existem obviamente prós e contras de uma arquitectura em relação à outra. O primeiro caso, tem a vantagem em relação ao segundo de apenas possuir um barramento, uma vez que só possuímos uma memória. No entanto, neste sistema existe uma maior limitação a nível de largura de banda, uma vez que apenas é possível adquirir um dos tipos de informação de cada vez. Esta arquitectura possui ainda o problema que do tamanho dos dados ter que ser do mesmo tamanho em bits que uma palavra do processador. A segunda arquitectura, possui a desvantagem de possuir mais um barramento para endereço e informação. No entanto, possui grandes vantagens o que leva que seja, actualmente, a arquitectura mais adoptada. Esta possui um *throughtput* mais elevado que a anterior, visto ser possível ler dados enquanto é requisitada a nova instrução, aumentando a velocidade de processamento. Possui igualmente a vantagem de o barramento para programa e dados não ter de possuir o mesmo tamanho, conseguindo deste modo uma maior flexibilidade a nível de arquitectura. [43]

3.3.1.1 *Instruction set*

Tendo em vista a construção de um processador, é necessário compreender como as instruções de um processador são construídas, de forma a, posteriormente, construir uma unidade de controlo e um datapath que implemente estas instruções.

O *instruction Set* é um assunto um pouco subjectivo, uma vez que este varia de arquitectura para arquitectura, apesar de em alguns pontos as diversas arquitecturas se cruzarem. Nesta secção irá ser abordado o caso do processador MIPS.

Independentemente da arquitectura usada, podemos dividir o *instruction set* de um processador em 5 tipos de instruções: **aritméticas**, **transferência de dados**, **lógicas**, **salto condicional** e **salto incondicional**.

- **Instruções aritmética**s – Visto uma das funções de um processador ser executar operações aritméticas, existe um conjunto de instruções que permite efectuar operações tais como: adição, subtracção etc.[14]

- **Instruções para transferência de dados** – Sendo os registos de uso geral muito limitados a nível de tamanho e de número, uma vez que apenas são utilizados para operações simples e passos intermédios, existe a necessidade de armazenar informação em memória (Ex: *array* de inteiros), pelo que é necessário aceder à memoria de dados. Estas instruções permitem, assim, escrever e ler valores da memória de dados.[14]

- **Instruções lógicas** – É muitas vezes útil ao programador efectuar operações lógicas tais como AND, OR, XOR, Shift, existindo desta forma este tipo de instruções.[14]

- **Instruções de salto condicional** – Muito raramente um programa segue uma ordem de execução ordenada, isto é, a instrução a ser executada é a seguinte em termos de posição de memória. As instruções de salto permitem, assim, saltar de uma posição de memória para outra definida pelo programador. As instruções de salto condicional, permitem saltar para uma pré-determinada posição de memória se uma condição se verificar. Ex: Salto para uma determinada posição de memória se um determinado registo de uso geral for zero.[14]

- **Instruções de salto incondicional** – Muito semelhantes às anteriores, mas sendo neste caso o salto efectuado sem que uma determinada condição seja satisfeita, apenas por necessidade do programador.

Para o caso da arquitectura MIPS, o correspondente *Instruction Set* reduzido é exibido na tabela seguinte. Existem outras instruções que não se encontram exibidas na tabela, não tendo interesse para a compreensão da arquitectura pelo que não serão exibidas.

A arquitectura exibida apresenta 32 registos de uso geral: $s0...$s7, $t0...$t9, $zero, $a0...$a3, $v0...$v1, $gp, $fp, $sp, $ra e $at. A descrição da sua função não é importante para a compreensão do trabalho.

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 - $s3	three register operands
Data transfer	load word	lw $s1,100($s2)	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	sw $s1,100($s2)	Memory[$s2 + 100] = $s1	Data from register to memory
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $$s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,L	if ($s1 == $s2) go to L	Equal test and branch
	branch on not equal	bne $s1,$s2,L	if ($s1 != $s2) go to L	Not equal test and branch
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; used with beq, bne
	set on less than immediate	slt $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than immediate; used with beq, bne
Unconditional jump	jump	j L	go to L	Jump to target address

Tabela 3.1: *Instruction Set* reduzido da arquitectura MIPS [14]

Observando a figura anterior, é fácil de verificar que nem todas as instruções possuem os mesmos nem o mesmo número de operandos. Este facto, leva a uma diferenciação do tipo de instruções. Na arquitectura MIPS distingue-se os tipos: R, I e J. A tabela seguinte mostra a correspondência entre o nome da instrução e o seu tipo.

MIPS instructions	Name	Format
add	add	R
subtract	sub	R
load word	lw	I
store word	sw	I
and	and	R
or	or	R
nor	nor	R
and immediate	andi	I
or immediate	ori	I
shift left logical	sll	R
shift right logical	srl	R
branch on equal	beq	I
branch on not equal	bne	I
set less than	slt	R
set less than immediate	slti	I
jump	j	J

Tabela 3.2: Correspondência entre o nome da instrução e o seu tipo [14]

A conversão das instruções *assembly* em instruções máquina será feita pelo designado *assembler*. Este, conforme o tipo de instrução, irá formar uma palavra binária sendo, esta sim, reconhecida pela máquina. As estruturas dos vários tipos de instrução serão agora apresentadas.[14]

Instrução do tipo R

A estrutura da instrução do tipo R encontra-se representada na figura seguinte.

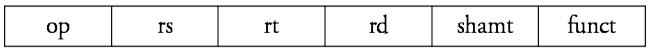

Figura 3.6: Estrutura da instrução do tipo R [14]

Observando a figura seguinte, distinguem-se os seguintes campos:

- **Opcode (Op) -** Indica a operação básica que deverá ser executada;
- **Rs -** Número do registo que contém o primeiro operando;
- **Rt -** Número do registo que contém o segundo operando;
- **Rd -** Número do registo que irá receber o resultado da operação;
- **Shamnt -** Serve para instruções de *shift* indicando o número de bits a serem shiftados;
- **funct -** Função a ser executada. Este campo, consiste numa variante da operação indicada pelo campo **op**. Ex: as operações ADD e SUB possuem o mesmo opcode, pelo que a distinção entre elas é feita através deste campo.

Instrução do tipo I

A estrutura do tipo I encontra-se representada na figura seguinte.

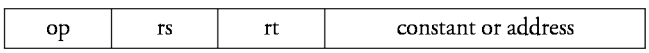

Figura 3.7: Estrutura da instrução do tipo I [14]

Este tipo de operação permite a atribuição de um valor imediato, ou seja, um valor que se encontra na própria instrução e não num registo. Este valor é limitado pelo número de bits do último campo. Os restantes campos possuem a mesma função que na instrução do tipo R.

Instrução do tipo J

A estrutura do tipo J encontra-se representada na figura3.8.

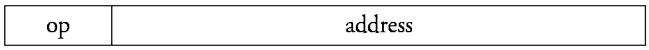

Figura 3.8: Estrutura da instrução do tipo J [14]

Este tipo de estrutura apenas é utilizado para a instrução *Jump*. Devido ao campo address possuir 26 bits para o endereço, é possível efectuar um salto maior em memória do que numa instrução do tipo I.

3.3.1.2 Modos de endereçamento

Tal como foi visto, para a execução de uma instrução existe a necessidade de aceder à informação que se encontra num registo ou em memória. Existe igualmente a necessidade de fazer saltos no programa (operações de salto). Os modos de endereçamento definem como a partir da informação das instruções os operandos são obtidos.[44]

Vamos de seguida abordar os diferentes modos de endereçamento que a arquitectura de exemplo possui.

Existem 5 formas de endereçamento: ***Immediate*, *Register*, *Base*, *Pc-relative*, *Pseudodirect***.

- ***Immediate* -** O valor do operando encontra-se na própria instrução.

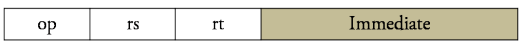

Figura 3.9: *Immediate Adressing* [14]

- ***Register* -** O operando encontra-se num registo.

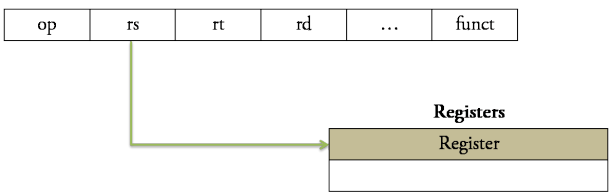

Figura 3.10: *Register Adressing* [14]

- ***Base*** - O operando encontra-se em memória, sendo a sua posição calculada através da soma de um registo com um valor indicado na própria instrução.

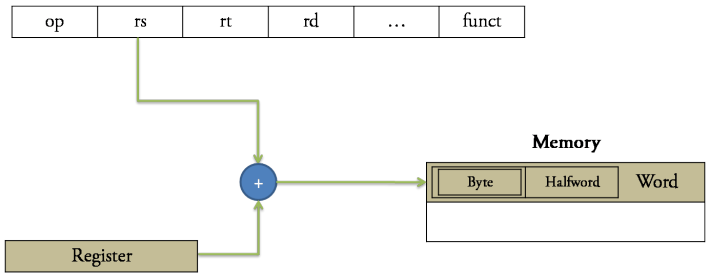

Figura 3.11: *Base addressing*[14]

- ***Pc-relative*** - Usado para instruções de salto incondicional. O endereço é calculado como sendo a soma entre o *Program Counter* mais um valor constante indicado na instrução.

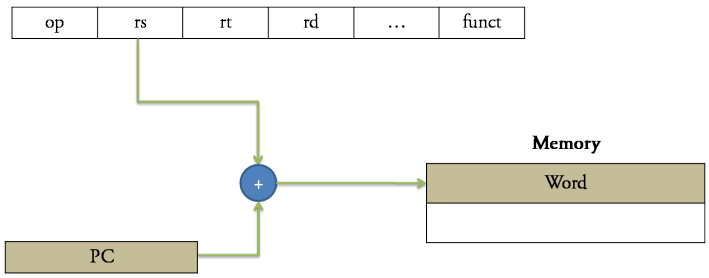

Figura 3.12: *Pc-relative addressing*[14]

- ***Pseudodirect*** - Usado para instruções de salto condicional, em que o endereço é calculado como sendo a concatenação de alguns bits mais significativos do *Program Counter* com o valor que se encontra na própria instrução.

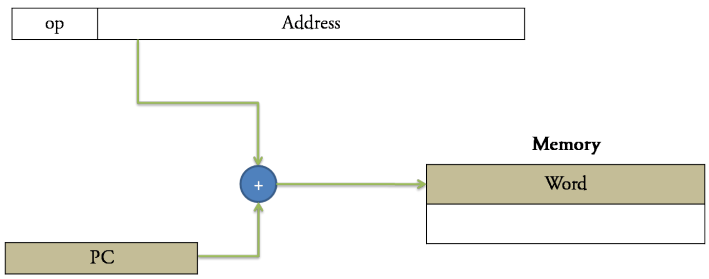

Figura 3.13: *Pseudodirect addressing*[14]

3.3.2 Processador

Até ao ponto actual foi visto a interface que um programador tem disponível para a elaboração de programas. Estes programas necessitam de algo físico que os execute sendo

esta é a função do microprocessador. Tal como foi visto, este divide-se, essencialmente, em dois componentes: ***datapath*** (Conjunto de unidades de execução) e **unidade de controlo**.

Qualquer que seja o microprocessador, podemos definir fluxo de operações que este executa a cada nova instrução: ***instruction fetch, instruction decode e execute.***

Figura 3.14: Fases de execução de uma instrução

- ***Instruction Fetch*** **-** Recolha da nova instrução para execução.

- ***Instruction Decode*** **-** Identifica a instrução e o seu tipo, por forma a que a unidade de controlo saiba quais os componentes que devem ser activos num determinado momento para a correcta execução da operação.

- ***Execute*** **-** A fase em que a instrução é executada e o resultado guardado.

Unidades de Execução e Unidade de controlo

As unidades de execução ou *datapath* permite a execução de uma determinada tarefa, enquanto a unidade de controlo permite o controlo da operação, isto é, a selecção dos componentes de forma a encaminhar a informação para obter um determinado resultado. O conjunto unidade de execução e *datapath* simplificado da arquitectura exemplo (MIPS) encontra-se representada na figura seguinte.

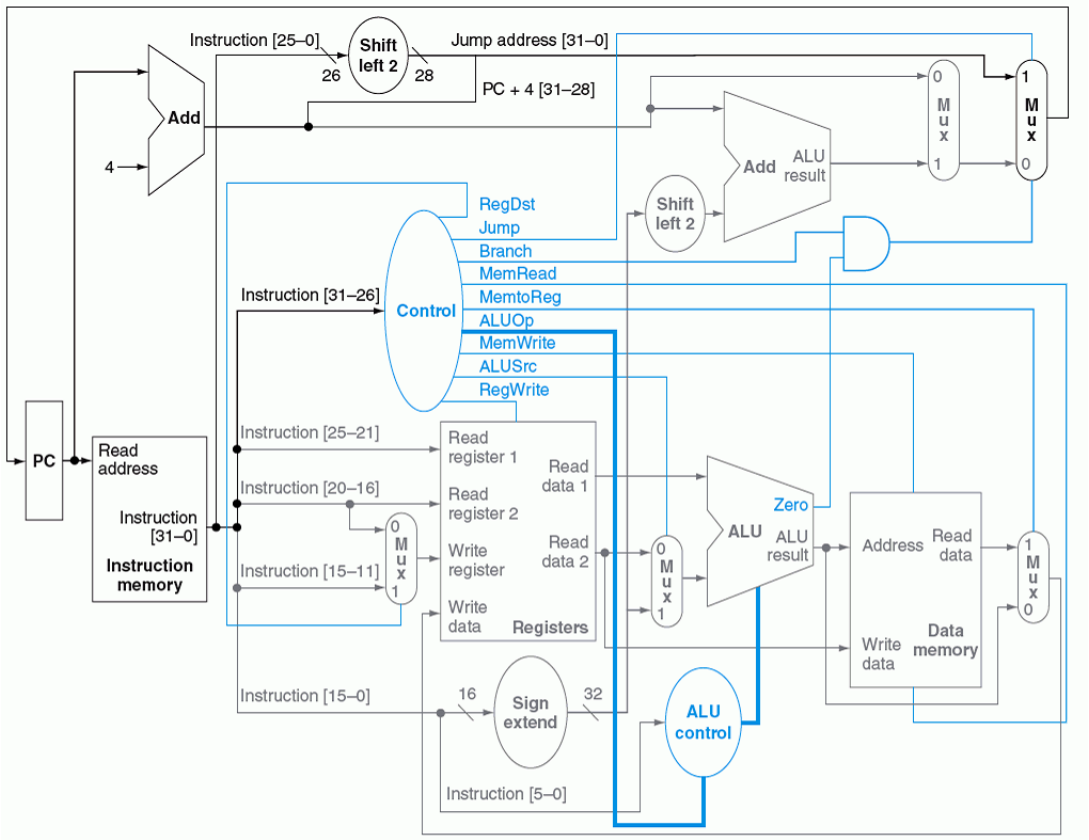

Figura 3.15: Exemplo de uma unidade de controlo e unidade de execução [14]

Observando a figura anterior, é possível verificar a existência de componentes aritméticos, memórias, *program counter*, portas lógicas, banco de registos etc, os quais são essenciais para execução de uma determinada instrução. É possível verificar igualmente que todos os componentes têm ligação com a unidade de controlo, visto estes serem controlados em qualquer instante por esta entidade, encaminhando-se, desta forma, a informação entre os diversos componentes. Observa-se igualmente que a unidade de controlo apenas possui uma entrada: os bits 31 a 26 da *instruction word*. Este corresponde ao *opcode* da instrução, constituindo a única informação necessária para que a unidade de controlo coordene os vários componentes. É possível verificar igualmente uma arquitectura do tipo ***harvard*** uma vez que, existe uma memória para armazenamento de dados e outra para o programa.

A estrutura apresentada tem a grande desvantagem da frequência máxima de funcionamento ser bastante baixa, uma vez que o sinal tem de passar por todos os componentes em apenas um ciclo de relógio (Arquitectura *Single-Cycle*). Para combater este problema apareceu a estrutura *Multi-cycle,* que permite executar uma instrução em vários ciclos de relógio, sendo que em cada ciclo uma tarefa simples é executada,

permitindo um aumento da frequência máxima do relógio. A sua estrutura não será apresentada.

Existe igualmente a estrutura *pipelined,* que divide as fases de *fetch*, *decode* e *execute*. Este processo é semelhante à linha de montagem de um automóvel, em que cada grupo se encontra responsável por uma determinada tarefa de montagem e em que a construção de um novo carro começa antes de o anterior terminar. O mesmo acontece nesta estrutura, em que são iniciadas as fases de uma instrução sem que a anterior tenha acabado, obtendo-se desta forma um maior *throughtput* de instruções.

3.3.3 Co-Processadores

Um processador de uso geral permite efectuar uma grande quantidade de operações. No entanto, existem operações muito específicas que este não é capaz de efectuar ou o tempo de execução que levaria seria muito elevado. Esta é a principal função de um co-processador, ou seja, fornecer ao processador principal, funções suplementares como por exemplo: operações em vírgula flutuante, gráficas, processamento de sinal, encriptação etc.[45]

3.4 Utilização de FPGAs para implementação de sistemas computacionais

A utilização de FPGAs para a implementação de processadores devido à sua grande flexibilidade tem sofrido nos últimos anos uma grande evolução devido ao aumento das capacidades lógicas destes dispositivos. É possível distinguir dois tipos de implementações de processadores em FPGA: ***Hard-Core*** e ***Soft-Core***.

No primeiro caso, o núcleo do processador encontra-se implementado originalmente na pastilha de silício da FPGA. A vantagem deste tipo de implementação encontra-se relacionada com a frequência de trabalho do processador a qual pode ser muito elevada. Assim, obtém-se um sistema computacional com um processador de grande velocidade e com periféricos que podem ser costumizados para uma determinada aplicação. A grande desvantagem desta implementação encontra-se na sua flexibilidade, uma vez que não é possível adicionar novas capacidades ao processador.

No segundo caso, o núcleo do processador é implementado inteiramente através de ferramentas de síntese, ou seja, é implementado através de ferramentas CAD. Devido a este facto, estes possuem a desvantagem de não conseguirem trabalhar à frequência bem como ter a *performance* da implementação *Hard-Core*. No entanto, estes permitem

uma grande flexibilidade, permitindo alterações significativas às capacidades do núcleo e adição de novas funcionalidades. [46]

3.4.1 Exemplos de microprocessadores em FPGA

De seguida são apresentados alguns dos microprocessadores disponibilizados pelo fabricante Xilinx.

PicoBlaze

Consiste num processador de 8 bits pertencente à categoria de *soft-cores* e foi construído para incorporação nas famílias Spartan 3, Virtex II e Virtex II Pro, oferecendo uma solução compacta para aplicações que não requerem uma grande quantidade de processamento de informação.[47]

MicroBlaze

Pertence igualmente à categoria de núcleos *soft-core* e consiste num processador RISP de 32 bits. Ao contrário do PicoBlaze, este possui já uma unidade de vírgula flutuante bem como outros componentes, que podem ser adicionados através de livrarias IP. É possível igualmente incorporá-lo na ferramenta EDK da Xilinx, permitindo que seja programado utilizando linguagens tais como C e C++.[48]

PowerPC440

O PowerPC440 foi criado pelo IBM, sendo mais tarde incorporado em FPGAs da Xilinx, pertencendo desta forma ao tipo *hard-core*. Este constituí um processador de 32 bits possuindo uma arquitectura *pipeline*, permitindo igualmente a utilização da ferramenta EDK de forma a facilitar a construção de aplicações.[49]

3.5 Trabalho Relacionado

Os trabalhos relacionados com o tema proposto nesta dissertação são muito escassos, sendo a abordagem ao problema feita de uma forma diferente. Nas arquitecturas consideradas como **VISC** (*Variable Instruction Set Computer*), existe um dicionário de instruções base e compete ao compilador escolher o melhor set de instruções a utilizar, tentando, desta forma, optimizar o código a compilar, permitindo uma execução mais eficiente do algoritmo. Existem igualmente processadores reconfiguráveis como por

exemplo: PRISM, PRISC, entre outros. Estes possibilitam a reconfiguração de um determinado hardware não permitindo, no entanto, a variação do *instruction set*.

O trabalho desta dissertação envolve igualmente uma componente de interacção remota para transferência de informação acerca das instruções a carregar, bem como um novo programa para um processador, não havendo igualmente informação disponível sobre este assunto.

3.6 Ferramentas de apoio

3.6.1 Xilinx Integrated Synthesis Environment (ISE)

O Xilinx ISE consiste numa ferramenta de *software* que permite a implementação, simulação e programação de qualquer FPGA ou CPLD deste fabricante. Todo o projecto é desenvolvido através de uma interface gráfica. A imagem seguinte mostra o aspecto desta ferramenta. Este ambiente de desenvolvimento encontra-se dividido basicamente em 5 blocos, numerados na figura 3.16 pelos números de 1 a 5.

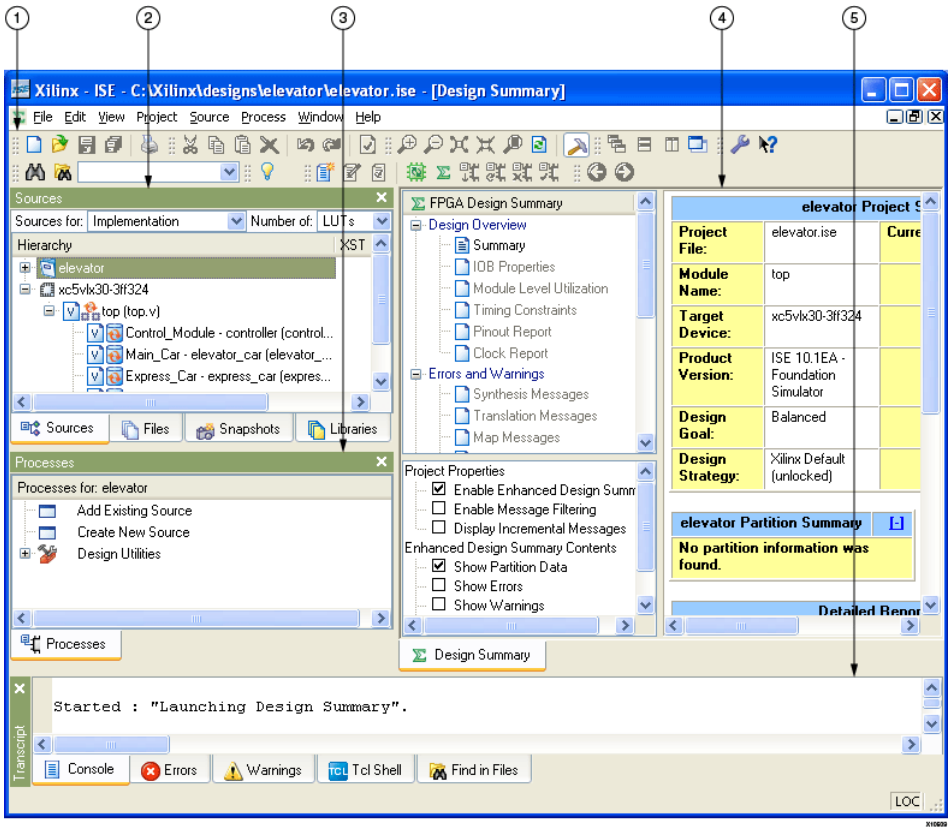

Figura 3.16: Ambiente da ferramenta ISE[15]

O bloco representado pelo número 1 representa a *Toolbar*, pelo que não é alvo de

grande explicação. Contém apenas alguns atalhos úteis para a implementação de um sistema digital.

O bloco 2 é designado por *Sources Window*. Esta janela constitui a primeira etapa para a criação de um projecto de dispositivo lógico programável. Mostra os ficheiros existentes no projecto, possibilitando igualmente a criação de novos ficheiros. Através da *tab sources* é possível observar uma hierarquia do projecto, isto é, todos os ficheiros que são utilizados para a implementação do projecto de uma forma hierarquizada. Observando a figura3.16, é possível observar que o módulo *Top* é o módulo superior do projecto, sendo que todos os restantes são chamados em camadas inferiores a estes. Ex: Main_Car (ver figura). Através deste bloco é também possível alterar definições do projecto, como por exemplo o modelo da FPGA, velocidade, *Package*.

O bloco 3 representa a *Process Window*. Este bloco permite ao *designer* implementar operações (processos) ao seu projecto. Os processos são exibidos igualmente de uma forma hierárquica. Este permite sintetizar, simular, implementar, criar módulos ou adicionar módulos já existentes. Dentro de cada processo existem ainda outros processos (forma hierárquica) encontrando-se segundo uma disposição que segue o diagrama de fluxo de um projecto. Este diagrama será apresentado num tópico posterior. A janela 4 representa o que mais usualmente se designa por "Área de trabalho". Nesta janela, é possível observar relatórios de implementação e outras informações úteis. É nesta janela que são editados os diversos ficheiros VHDL, Verilog, Esquemático etc. Por último (janela número 5), encontra-se a janela *Trasncript Window,* que permite ao utilizador observar os processos correntes, bem como a visualização de mensagens importantes, tais como erros e avisos. Permite igualmente, em caso de erro de sintaxe, o redireccionamento para a linha de código correspondente ou, sendo um erro de síntese, a implementação permite o redireccionamento para a internet para possíveis soluções.[15]

Etapas de um projecto utilizando Xilinx ISE

Para a implementação de um projecto para FPGA ou CPLD é necessário seguir determinados passos sequencialmente. O diagrama de operações seguido na ferramenta ISE encontra-se representado na figura3.17:

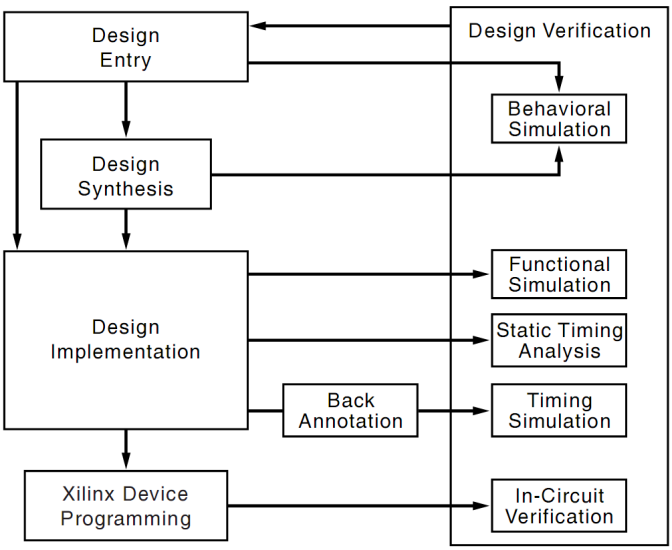

Figura 3.17: Diagrama de Fluxo de um projecto na ferramenta ISE[16]

- ***Design Entry*** ou criação do projecto – Começa-se pela criação de um projecto. Neste, serão incluídos ficheiros VHDL, Verilog, do tipo esquemático, blocos das livrarias IP e ficheiros UCF. Os ficheiros UCF representam as limitações a nível de ligação de pinos, de tempo e de área impostas pelo designer. Depois da criação do projecto, poder-se-á passar para a fase de síntese ou para a fase de simulação comportamental.

- ***Design Synthesis*** ou síntese do projecto – Depois da implementação efectuada, é necessário sintetizar o sistema desenvolvido. Primeiro é iniciado um processo de detecção de erros de sintaxe. De seguida, são detectados erros de implementação. Ex: um sinal alterado por processos diferentes. Quando todos os conflitos estiverem resolvidos, é criado um ficheiro designado por *netlist* que servirá de entrada para a etapa *Design Implementation*.

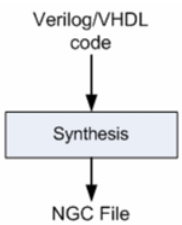

Figura 3.18: Entrada/Saída do processo de Síntese

- ***Design Implementation*** ou implementação do projecto – Nesta fase, o ficheiro criado pela síntese será convertido num projecto lógico que irá criar um ficheiro

físico, que pode ser depois enviado para uma FPGA ou CPLD. Esta fase pode ser dividida em três subfases: *Translate*, *Map* e *Place and Route*. Na fase *Translate* as indicações contidas no ficheiro UCF são convertidas para um ficheiro lógico.

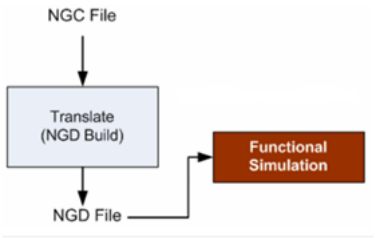

Figura 3.19: Entrada/Saída do processo de *Translate*

A seguir, é executada a fase *Map*. Nesta, o circuito lógico criado é dividido, de forma a ser enviado para FPGA, ou seja, todo o circuito é dividido pelos CLB e IOB disponíveis.

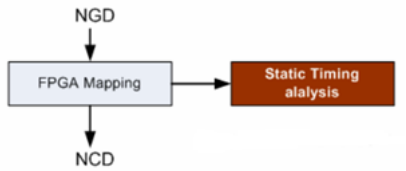

Figura 3.20: Entrada/Saída do processo de *Map*

Por fim, a fase *Place and Route*, cria as ligações entre os vários blocos CLB ou IOB, formando o sistema construído, distribuído pela os diversos componentes lógicos.

- ***Xilinx Device Programming*** ou programação do dispositivo lógico – Chegando a esta fase, o projecto deverá ser carregado para a FPGA. No entanto, antes do envio, o ficheiro produzido até ao momento deverá ser convertido num bistream, para que a FPGA ou CPLD o aceitem: .bit. Depois de criado o ficheiro .bit, é necessário programar a FPGA através do aplicativo incorporado no ISE, iMPACT e um cabo de programação. A maior parte das placas de desenvolvimento estas possuem um software adicional que permite programar usualmente por ligação USB.

- ***Design Verification*** ou teste do projecto – a verificação de um projecto pode decorrer em diferentes etapas da construção de um sistema. Podem distinguir-se os principais momentos da verificação de um projecto:

- ***Behavioral Simulation*** ou simulação comportamental – Esta constitui a primeira simulação do sistema construído e pretende verificar se a implementação efectuada corresponde em termos ideais ao pretendido. Nesta simulação, é permitido simular código VHDL ou Verilog através de simuladores que conseguem exibir o valor de todos os sinais num determinado instante de tempo. A ferramenta ISE possui já um simulador: *ISE Simulator*. Em tópico posterior, iremos fazer uma breve apresentação ao simulador *ModelSim*.
- ***Functional Simulation*** ou simulação funcional – A simulação funcional é feita depois da fase *Translate*, que fornece informação sobre a operação lógica do circuito implementado. O projectista pode verificar, neste ponto, a funcionalidade do seu projecto. Caso esta não se encontre como o esperado, é necessário alterar o código HDL.
- ***Static Time Analysis*** ou Analise temporal – Depois da fase *Map* e *Place and Route* são criados relatórios que fornecem informações sobre os atrasos dos sinais que são entregues ao circuito desenvolvido. Estes relatórios são bastante importantes, uma vez que é possível por exemplo verificar a frequência máxima de operação do circuito implementado.
- ***Timing Simulation*** ou Simulação Temporal – A simulação temporal permite analisar o sistema considerando já os atrasos existentes nos sinais. Esta simulação pode ser feita pelos mesmos simuladores utilizados para a simulação comportamental, sendo, neste caso, uma simulação mais fiável do que na simulação comportamental.
- ***In-Circuit Verification*** ou Verificação no circuito – Constitui a última forma de verificar a funcionalidade do circuito. Após carregar a FPGA com uma determinada implementação, é possível ainda verificar o comportamento do sistema a nível físico, isto é, pode-se capturar todos os sinais necessários para que possam ser enviados para o computador e serem analisados. A ferramenta para fazer este debug in circuit designa-se *ChipScopePro*.

3.6.2 Xilinx Core Generator

Esta ferramenta encontra-se incluída no ISE e permite a construção de blocos optimizados para FPGA. Estes blocos encontram-se em livrarias designadas por Livrarias IP (*Intelectual Property*). O ISE inclui já uma livraria (*LOGICore*) construída pela própria Xilinx, possuindo componentes que abrangem uma vasta área de aplicações e que

são de uso livre. É possível, igualmente, através de internet, obter mais componentes para as livrarias IP.

As áreas que a livraria LOGICore abrange são:

- **Automóvel e industrial** – Este é constituído essencialmente por Protocolos de Comunicação, frequentemente utilizados nestas áreas. Ex: CAN.
- **Componentes Básicos** – Bloco possuindo comparadores, *Shift Registers*, *Flip-Flops* etc. Estes componentes encontram-se devidamente optimizados, sendo, por isso, útil a sua utilização.
- **Comunicações e Redes de Telecomunicações** – Estes blocos permitem efectuar processamento digital a nível das comunicações, com fios e sem fios. Possui correctores de erros (Ex: Descodificador de Viterbi), blocos *Ethernet*, LVDS (*Low-voltage differential signaling*), Modulação (Ex: *Direct Digital Synthesizer*), Interfaces Serie (Ex: RS-232).
- **Debug** – Possui ferramentas para debug do sistema. (Ex: *Chip Scope Pro*).
- **Processamento Digital de Sinal** – Possui blocos para correlação de sinais, filtros (Ex: FIR), Multiplicadores, Transformadas (Ex: DFT), funções trigonométricas etc.
- **Funções Matemáticas** – Possui componentes que permitem efectuar operações em vírgula flutuante, Multiplicadores, raiz quadrada, divisão, etc.
- **Elementos de Memória** – Este grupo contém memórias CAM (*Content-addressable memory*), memórias RAM e ROM as quais se encontram optimizadas e elementos de ordenação tais como FIFOs.
- **Interfaces para barramentos** – Possui componentes de interface com barramentos tais como PCI, PCI Express e RapidIO.
- **Processamento de áudio e vídeo** – Possui componentes para processamento digital de áudio e vídeo tais como aceleradores gráficos, correctores gamma, codificadores H.264, codificadores e descodificadores MPEG etc.

3.6.3 ModelSim

A ferramenta ModelSim é uma ferramenta poderosa para desenvolvimento de sistemas para FPGAs que permite a criação, gestão, simulação e implementação de sistemas

digitais, facilitando o design e verificação de uma determinada função de forma simples e rápida.

Permite ao projectista utilizar diferentes métodos e linguagens lógicas no mesmo projecto. Este pode incluir as linguagens VHDL e Verilog. Os blocos desenvolvidos podem ser, depois, transformados em diagramas de blocos, permitindo a ligação entre vários blocos através de uma interface gráfica. É possível igualmente construir blocos que implementam um determinado fluxo de operações (máquina de estados finitos) graficamente. A figura3.21 mostra o exemplo de um diagrama de blocos e de uma máquina de estados finitos criados graficamente. [50]

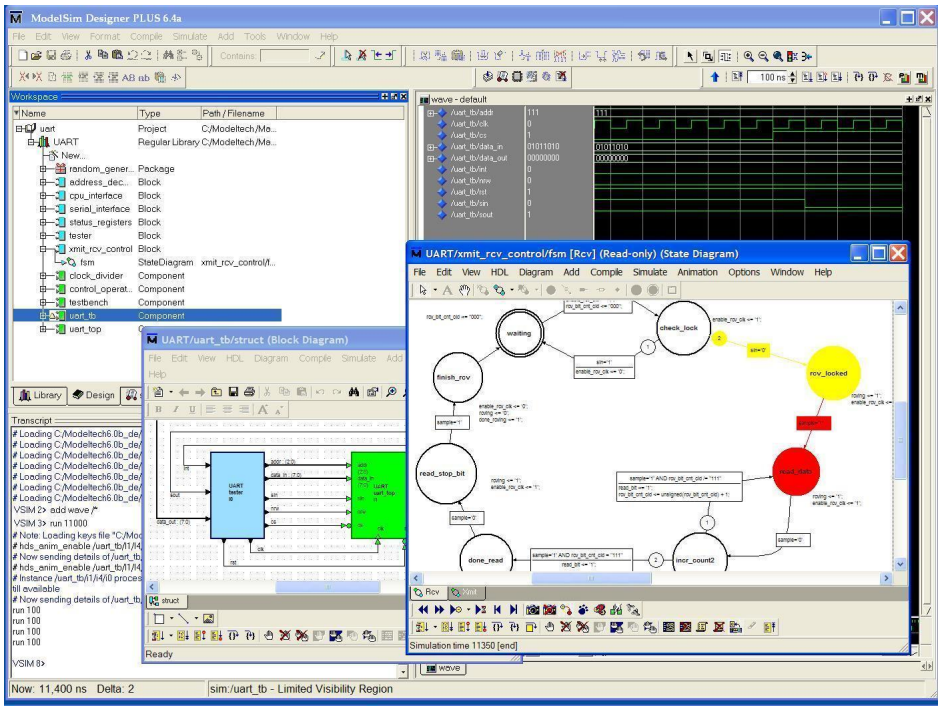

Figura 3.21: Ambiente de desenvolvimento da ferramenta ModelSim [17]

A ferramenta ModelSim permite, igualmente, a passagem destes blocos implementados graficamente para uma das linguagens de descrição de hardware, conseguindo-se, deste modo, passar facilmente um projecto em modelsim para qualquer outro editor.

É possível verificar, a qualquer momento, com o ModelSim o comportamento de qualquer componente, separadamente ou em conjuntos com outros componentes, através de diagramas temporais. Uma vez que esta ferramenta permite a integração de ferramentas de síntese e de implementação (Synplicity, Xilinx ISE e Altera Quartus), é possível elaborar simulações tendo em conta as limitações do projecto, tais como atrasos dos componentes, frequência máxima de trabalho etc, conseguindo-se, desta forma, uma simulação mais próxima da realidade.[51]

3.6.4 Eagle

Visto as placas de desenvolvimento não possuírem um módulo wireless integrado, foi necessário construir placas de circuito impresso para adaptar o módulo às respectivas placas. Para tal foi utilizada a ferramenta Eagle que permitiu igualmente a construção de circuitos impressos para fazer *degub* dos sinais provenientes do módulo CC1101, de que falaremos em tópico posterior. Esta ferramenta, possui a facilidade de incorporar uma grande quantidade de componentes e conectores, permitindo a rápida construção de uma placa de circuito impresso. A sua interface encontra-se representada na figura seguinte.

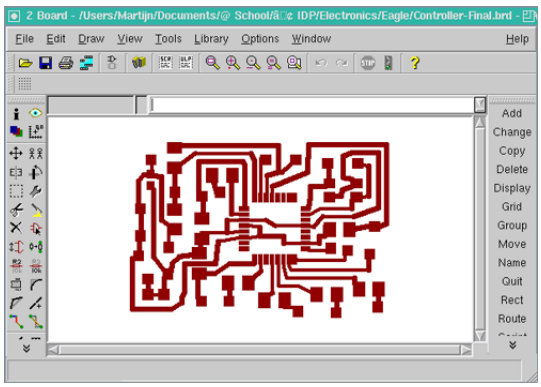

Figura 3.22: Ferramenta *Eagle* [18]

3.7 Placas de desenvolvimento

Para a execução deste trabalho foram utilizadas duas placas de prototipagem em FPGAs diferentes: **Celoxia RC10** e **Nexys 2**.

Celoxica RC10

Esta placa de prototipagem foi desenvolvida pelo construtor Celoxica. Um diagrama de blocos representativo desta board encontra-se na figura seguinte.

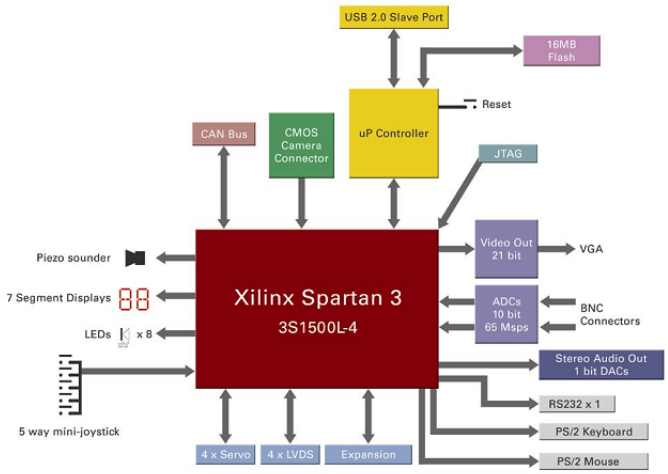

Figura 3.23: *Overview* da placa de desenvolvimento Celoxica RC10 [19]

Principais características

- Xilinx Spartan 3L XC3S1500L-4-FG320;
- Micro *joystick* com 5 posições;
- Porta PS/2 para rato e teclado;
- Porta RS-232;
- 2 conversores analógico-digitais de 10 bits;
- Saída para monitor VGA
- Saída para LCD;
- Duas saídas de áudio;
- Um microcontrolador com ligação USB 2.0 para configuração da FPGA e para acesso a uma memória *flash* de 16MB;
- 2 displays de sete segmentos;
- 8 LEDs;
- Um conector de expansão com 50 pinos o qual inclui pinos de alimentação, de I/O e de *clock*;
- Ligação até 4 servo motores;
- Conector para barramento CAN;
- Conector JTAG;

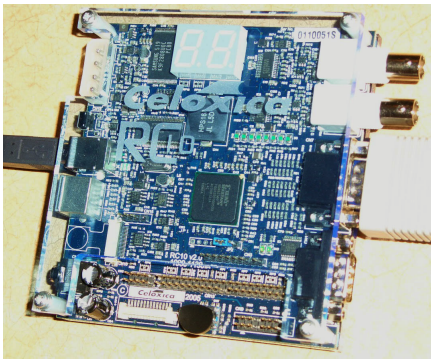

Figura 3.24: Celoxica RC 10 [20]

Nexys 2

Esta placa foi desenvolida pelo fabricante Digilent. A figura seguinte mostra um diagrama representativo desta.

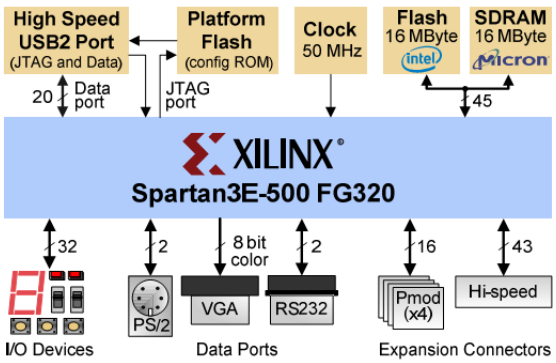

Figura 3.25: *Overview* da placa de desenvolvimento Nexys 2 [21]

Principais características

- Xilinx Spartan 3E-500 -4 -FG320;
- Porta USB 2.0 para transferência de dados e configuração da FPGA;
- Alimentação através do USB, Baterias ou transformador;
- 1 Memória *flash* da Intel com capacidade de 16Mbyte;
- 1 Memória SDRAM da Micron com capacidade de 16Mbyte;
- Oscilador de 50MHz e capacidade de um segundo oscilador;
- 60 pinos I/O através de 5 conectores de expansão;
- 8 LEDs;

- 4 digitos de 7 segmentos;
- 8 *switches*;
- 4 botões;

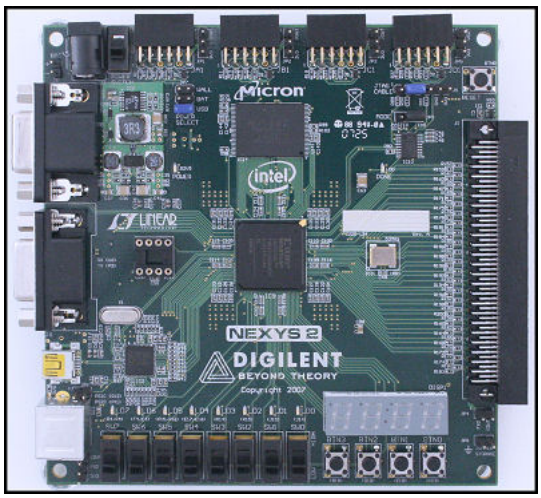

Figura 3.26: Placa de desenvolvimento Nexys 2 [22]

Capítulo 4

Processador com um *instruction set* variável

4.1 Sumário

Pretende-se neste capítulo apresentar o processador com o *instruction set* variável desenvolvido.

Começa com uma introdução à ideia que presidiu à realização deste processador. Por forma a conseguir a característica de reconfigurabilidade da unidade de controlo, é apresentada uma máquina de estados finitos reprogramável, a qual desempenha uma função decisiva para a construção do componente proposto. De seguida, é apresentado o processador desenvolvido bem como a explicação de todos os componentes necessários à obtenção de um *instruction set* variável. Para finalizar, é apresentado o processador de uso geral ao qual foi ligado o processador construído sendo apresentados alguns resultados sobre o sistema desenvolvido.

4.2 Introdução

Tal como foi apresentado no capítulo 3, um processador pode dividir-se em dois principais componentes: ***datapath*** e **unidade de controlo**. Estas duas unidades, em conjunto, permitem-nos executar instruções.

Ideia base

Uma instrução segue sempre um determinado fluxo de operações. Estas são executadas pelo *datapath,* enquanto o controlo e encaminhamento dos diversos sinais é da respon-

sabilidade da unidade de controlo. Esta unidade de controlo segue, desta forma, um diagrama de fluxo de sinal.

Para cada instrução este diagrama de fluxo de sinal é diferente. Tendo em conta este facto, pode-se definir um processador com *instruction set* variável como um processador em que unidade de controlo possui um fluxo de sinal variável, o que permite a execução de fluxo de operações diferentes, ou seja, instruções diferentes.

A unidade de controlo de um processador é geralmente feita recorrendo a uma máquina de estados finitos. Esta, para cada instrução segue determinados passos, que por sua vez, são comuns em parte a outras instruções.

Desta forma, para construção de um processador com *instruction set* variável, é necessário a construção de uma máquina de estados finitos em que o seu fluxo de operações pode ser alterado, contendo apenas a informação sobre uma determinada instrução (Máquina de estados finitos reprogramável), podendo esta ser alterada. Em relação ao *datapath*, este terá de ser fixo e o mais genérico possível de forma a acomodar o maior número de instruções possíveis. Este facto leva a uma maior complexidade do *datapath*.

O processador construído tem como objectivo a execução de operações sobre vectores binários e ternários, permitindo criar as bases para um futuro processador de uso geral.

4.3 Máquina de estados finitos

4.3.1 Máquina de estados finitos simples

Uma máquina de estados finitos (FSM) consiste numa abstracção utilizada em sistemas digitais que permite impor alguma sequência de operações, sendo constituída por uma série de entradas, saídas e estados. A transição entre estados é feita de acordo com uma função de transição, enquanto as saídas são governadas por uma função de saída. O conceito de **estados** é abstracto, permitindo apenas marcar a sequência de operações.Em termos práticos um estado consiste numa variável que possui um número de valores finitos, o que explica a designação de "finitos".

Uma FSM possui informação sobre o estado corrente bem como o estado para o qual irá saltar na próxima transição do relógio, pelo que esta estrutura possui um componente de memória. O próximo estado é calculado tendo em conta a já dita **função de transição**.

Este tipo de estrutura tem duas variantes: **Máquina de Mealy** e **Máquina de Moore** que definem como a saída é afectada pela entrada. Na primeira variante, a

saída depende, para além do estado corrente, do valor da entrada nesse mesmo instante. No segundo caso, a saída apenas depende do estado corrente.[52] As figuras 4.1 e 4.2 mostram cada um dos casos.

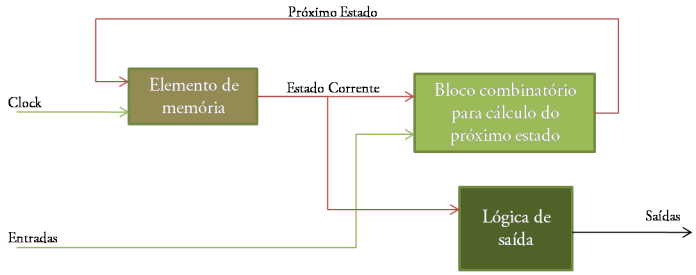

Figura 4.1: Máquina de Moore

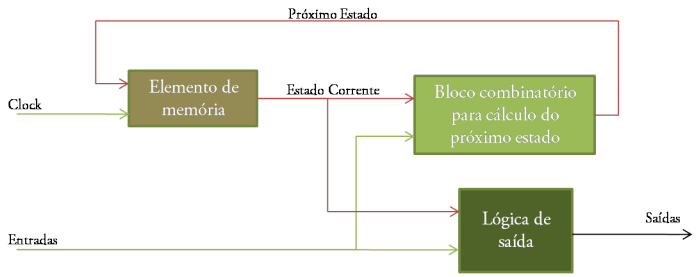

Figura 4.2: Máquina de Mealy

4.3.2 Máquina de estados finitos reprogramável

Uma máquina de estados finitos reprogramável (RFSM) consiste numa máquina de estados finitos em que é possível a alteração do seu comportamento. Este pode ser feito de duas maneiras: a primeira abordagem, consiste na utilização de FPGAs que têm a capacidade da reconfiguração parcial, ou seja, alterar fisicamente parte da FPGA. A segunda abordagem, consiste na utilização de memórias RAM e ROM. Uma vez que se torna mais fácil e nem todas as FPGAs permitem a reconfiguração parcial, foi adoptado o segundo método. Uma análise mais aprofundada deste tipo de estrutura pode ser vista em [23]

Implementação

Tal como foi dito, uma máquina de estados finitos reprogramável pode ser feita com base em memórias. A figura seguinte, mostra um primeiro esquema de como esta pode ser construída.

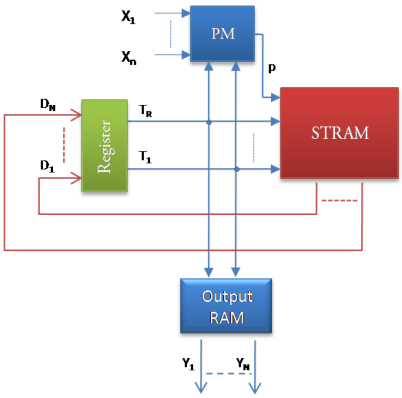

Figura 4.3: Máquina de estados Finitos reprogramável [23]

Esta máquina possui os seguintes componentes:

- **Registo** (*Register*) - Permite armazenar o valor do estado corrente da RFSM;
- ***Multiplexer* Programável** (PM) - Permite seleccionar a entrada pretendida conforme as especificações da máquina. É constituído por uma memória RAM (*Multiplexer RAM*) e um *multiplexer*, sendo as entradas de selecção deste as saídas da memória RAM. Este componente encontra-se representado na figura seguinte;

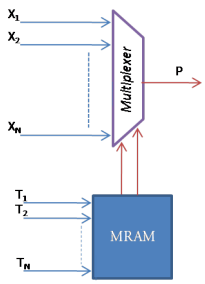

Figura 4.4: *Multiplexer* Programável (PM) [23]

- **SRAM** (*State* RAM) - Permite através do estado corrente e da entrada selecionada pelo bloco PM calcular o estado seguinte;
- **Memória de saída (*Output* RAM) -** Esta permite através do estado corrente calcular a saída da máquina;

No entanto, esta máquina de estados apresenta um problema, uma vez que, quando existe a necessidade de avaliar mais do que uma entrada, será necessário a existência

de estados adicionais, aumentado assim o tempo de execução da máquina, uma vez que só uma entrada em cada ciclo de relógio poderá ser analisada. Vejamos um exemplo:

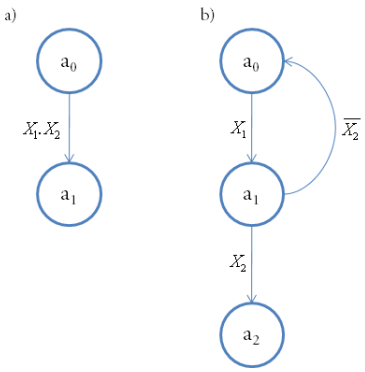

Figura 4.5: a) Diagrama de estados b) Diagrama de estados equivalente na RFSM

Observando a figura 4.5, verifica-se que o número de estados aumenta proporcionalmente com o número de entradas, ou seja, se é necessário avaliar n entradas, vamos necessitar de n-1 estados adicionais. Assim, se o número de entradas for relativamente elevado, o processamento irá tornar-se muito pouco eficaz.

Para resolver este problema surge um modelo baseado em ***Dummy States***. *Dummy States* são estados fictícios que influenciam muito pouco a velocidade de transição entre estados. Esta encontra-se limitada, basicamente, pelo tempo de propagação pelos vários componentes.[23]

A figura seguinte mostra o diagrama de estados correspondente:

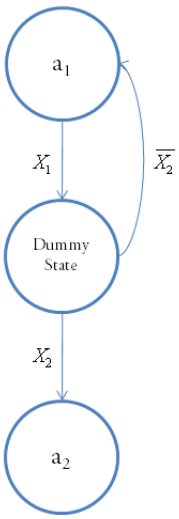

Figura 4.6: Diagrama de estados com *Dummy States*

A sua implementação com máquinas de estados finitos encontra-se na figura seguinte. Esta máquina designa-se por RFSM em cascata.

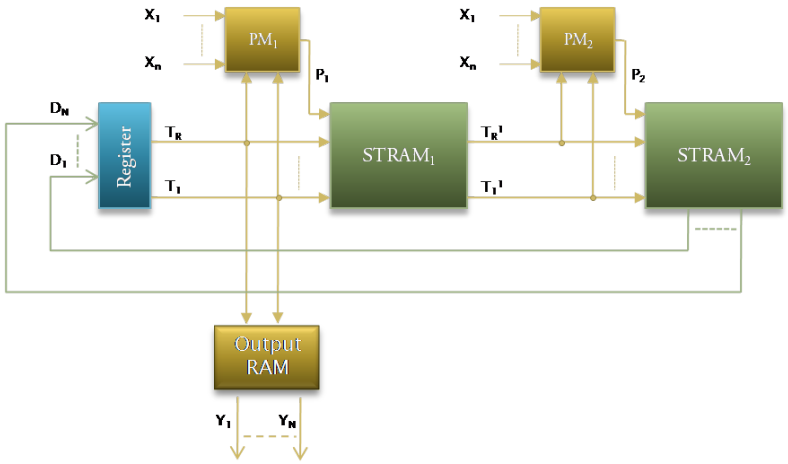

Figura 4.7: RFSM em cascata [23]

4.4 Processador Combinatório com um *instruction set* variável

Com este trabalho pretendeu-se implementar um processador com um *instruction set* variável, ou seja, que permita reprogramação, de forma a executar uma instrução diferente caso seja necessário. Esta reprogramação deverá ser feita em *run-time*, isto é, deverá ser possível mudar o *instruction set* do processador sem que seja necessário carregar um novo ficheiro de configuração na FPGA. Este processador permite fazer operações sobre vectores binários e ternários.

O processador desenvolvido baseia-se na utilização de máquinas de estados finitos reconfiguráveis (RFSM), permitindo seleccionar blocos de um *datapath* que irá executar funções auxiliares para a concretização da operação em causa. Deste modo, o *datapath* deverá ser o mais generalista possível para conseguir executar uma vasta gama de instruções.

4.4.1 Macro - Esquema do sistema desenvolvido

Na figura seguinte é apresentado um Macro - Esquema do sistema desenvolvido:

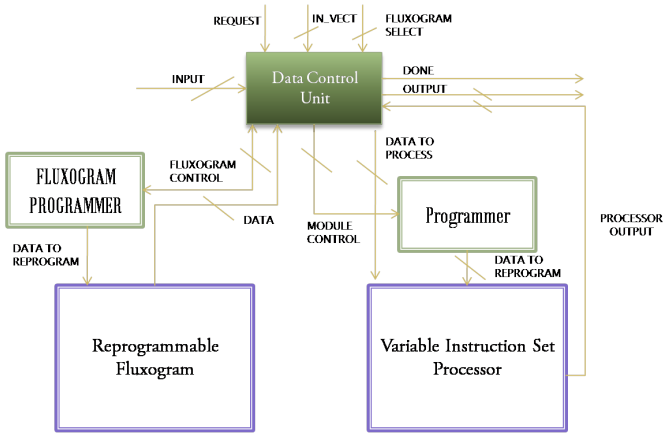

Figura 4.8: Sistema desenvolvido

Observando a Figura 4.8 verifica-se que existem 5 blocos principais: Uma unidade de controlo, dois módulos reconfiguráveis (***Variable Instruction Set Processor*** e ***Reprogramable Fluxogram***) e dois programadores das unidades reconfiguráveis. A descrição destes blocos será feita nas secções seguintes.

4.4.2 *Variable Instruction Set Processor*

Este bloco consiste numa máquina de estados finitosreprogramável, um *datapath*, um componente de controlo e um conversor paralelo-serie.O macro - esquema seguinte ilustra este bloco:

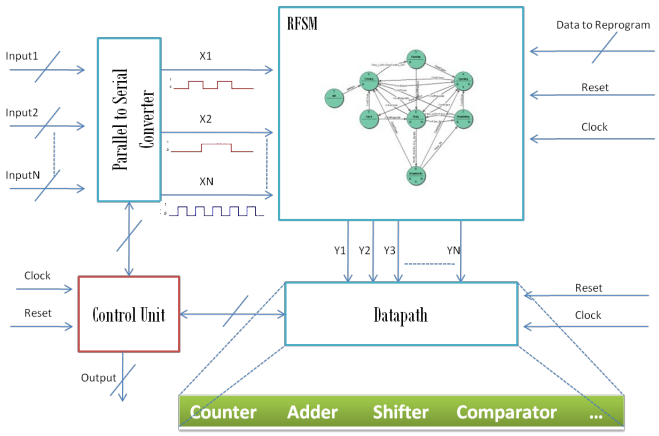

Figura 4.9: *Variable Instruction Set Processor*

- ***Parallel to serial Converter*** - Permite receber um determinado vector e transformá-lo numa sequência série. Este bloco é muito importante, uma vez que, se a informação entrasse na RFSM em formato paralelo o número de estados necessários seria enorme.

- **RFSM (unidade de controlo principal)** - Permite executar um fluxo de operações (reprogramável) de forma a concretizar a operação pretendida. Para a execução de uma determinada operação (instrução) as saídas da RFSM, encontram-se ligadas a unidades de execução do *datapath*,de forma a efectuar algumas operações básicas tais como: contar, somar etc.

- **Control Unit** - Permite efectuar algumas operações de controlo adicionais sendo estas controladas pela saída da RFSM (unidade de controlo principal). Recebe igualmente o resultado da execução de uma instrução e reencaminha-a para o exterior.

- ***Datapath*** - Permite a implementação das operações aritméticas que levam a um resultado. O controlo de quais os blocos deste que deverão ser adicionados, encaminhando assim o resultado por um caminho específico é feito através das saídas da RFSM.

A figura seguinte apresenta os diversos componentes de uma forma mais detalhada que constituem o processador desenvolvido. É de notar que o componente *Control Unit* e *Parallel to serial Converter* se encontram apresentados na figura 4.10 como um único componente.

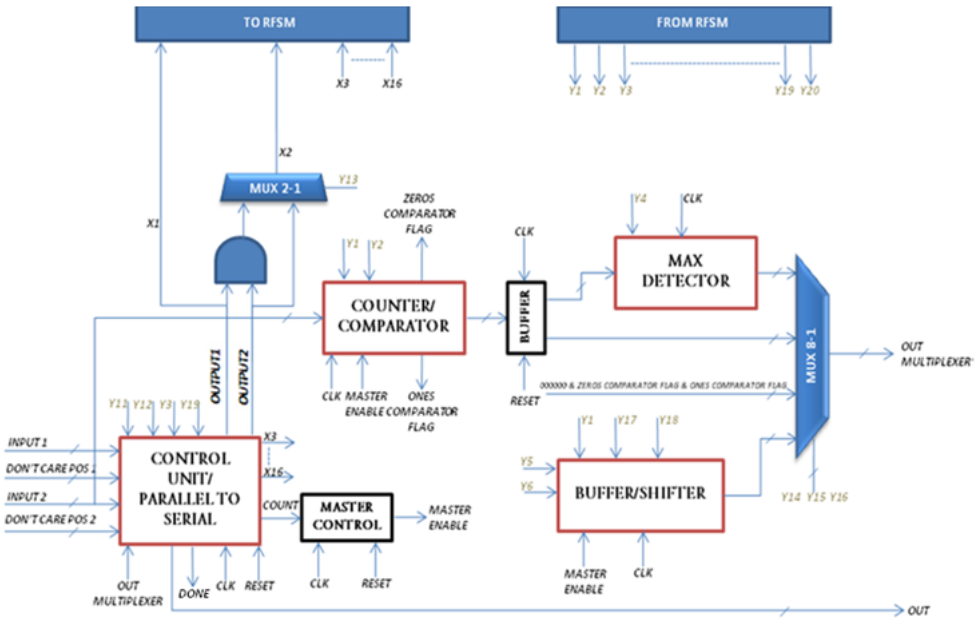

Figura 4.10: *Variable Instruction Set Processor*

Observando a figura datapath, podemos distinguir dois conjuntos de blocos: **Unidades de Controlo** e **Unidades de Execução**.

As Unidades de Controlo, tal como o nome indica, são componentes que permitem a correcta execução da operação pretendida. Estas controlam sinais para que a saída da operação seja encaminhada correctamente. Pelo diagrama anterior, verifica-se a existência de três unidades de controlo: ***Controlo Unit/Parallel to Serial***, ***Master Control*** e **RFSM**. As unidades de execução são os componentes que permitem efectuar uma determinada operação aritmética. Estes são: ***Counter/Comparator***, ***Max Detector*** e ***Buffer/Shifter***. O ***Buffer*** que se encontra entre o *Counter/Comparator* e *Max Detecto*r, não se encontra englobado em alguma destas categorias. No entanto, este é um elemento essencial para o correcto funcionamento do *datapath*. A sua função será discutida mais à frente. Na Figura 4.10, é de notar que os componentes com moldagem a cor vermelha são actualizados na transição descendente do relógio e os componentes de cor preta na transição ascendente do relógio. Todos os restantes componentes, excepto a RFSM que actualiza o seu estado na transição descendente, não são actualizados com o sinal de relógio.

Analisemos então o comportamento de cada um dos componentes apresentados.

4.4.2.1 Unidades de Execução

Counter/Comparator

Este bloco, tal como o nome indica, permite contar e comparar valores. Este componente possui cinco entradas: **Y1**, **Y2**, ***MasterEnable***, ***Input2*** e ***Clock***.

- **Y1** - Este sinal provêm da RFSM e permite fazer um *Reset* ao contador isto é, colocar o contador a zero. É activo alto.

- **Y2** - Este sinal provém igualmente da RFSM e permite activar a contagem, isto é, quando este sinal se encontra com o valor lógico '1', na transiçãodescendente do relógio o contador interno incrementa uma unidade.

- ***MasterEnable*** - Este sinal provêm do bloco *Master Control*. Quando este se encontra a '1', o bloco *Counter/Comparator* encontra-se activo, caso contrário, inactivo, isto é, mesmo que os restantes sinais de entrada variem, o componente não actualiza os seus valores.

- ***Input2*** - Este sinal constitui um valor de comparação.

- ***Clock*** - Simples sinal de relógio. Neste componente, o valor de saída é actualizado na transição descendente.

Este apresenta três saídas: **Valor do Contador**, ***Zeros Comparator Flag*** e ***Ones Comparator Flag***.

- **Valor do Contador** - Possui o valor actual do contador. Este valor é actualizado na transição descendente do relógio.
- ***Zeros Comparator Flag*** - Este sinal indica se o valor actual da subtracção do tamanho dos vectores a analisar e o contador possui o valor indicado pela entrada *Input2*, tomando o valor '1' em caso afirmativo.
- ***Ones Comparator Flag*** - Este sinal indica se o valor actual do contador é igual ao indicado pela entrada *Input2*.

Max Detector

Este bloco permite detectar qual o vector máximo que foi detectado desde o último *reset*. Este possui apenas duas entradas: **Y4** e **Clock**:

- **Y4** - Permite efectuar um *reset* do valor de máximo. Este sinal provém da máquina de estados finitos reprogramável.
- **Clock** - Sinal de relógio. Este componente é actualizado na transição descendente do relógio.

Este apresenta apenas uma saída: **Valor de Máximo**. Esta indica o máximo valor encontrado na sua entrada que, no nosso caso, é o valor proveniente do contador.

Buffer/Shifter

Para algumas operações pode ser mais fácil utilizar directamente a saída da RFSM. Para isso, foi criado o bloco *Buffer/Shifter*. Este, permite passar a informação das saídas Y5 e Y6 directamente para a saída (*Buffer*), através da concatenação com zeros, de forma a obter um vector com o tamanho da saída genérica ou então ir fazendo um shift (*shifter*), para que os sucessivos valores fornecidos pela saída Y5 sejam convertidos de informação em série para paralelo. Este bloco possuiu sete entradas: **Y1**, **Y5**, **Y6**, **Y17**, **Y18**, **Clock** e **MasterEnable**:

- **Y1** - Este sinal proveniente da RFSM quando com o valor lógico '1' coloca a saída a zero, sendo todos os valores recebidos até ao instante descartados.
- **Y5**, **Y6** - Sinais que provêm da RFSM que irão ser colocados na saída (Y5 e Y6) ou shiftados (apenas Y5).

- **Y17** - Sinal que, tomando o valor lógico '1', permite efectuar uma das duas operações.

- **Y18** - Permite seleccionar a operação desejada. Quando como o valor lógico '1' a operação *Shift* é seleccionada. Caso contrário, a operação escolhida desempenha uma função de *Buffer*.

- **Clock** - Sinal de relógio. Os sinais de saída são actualizados na transição descendente do relógio.

- **MasterEnable** - Possui a mesma função que para o bloco *Counter/Comparator*. Provêm do bloco Master Control e só quando este se encontra activo os sinais internos são actualizados

Em relação às saídas, este apenas possuí uma saída contendo o valor do vector em que se encontra armazenada a informação. Este vector de saída pode conter informação *shiftada* ou, simplesmente, os valores de Y5 e Y6 concatenado com zeros.

4.4.2.2 Unidades de Controlo

As unidades de controlo permitem controlar o fluxo de operações no *datapath*. Como foi referido anteriormente, este é constituído por três componentes: **RFSM**, **Control Unit / Parallel to Serial** e **Master Control**. O bloco **RFSM** já foi explicado em secção anterior, sendo apenas de realçar que este possuí possui 16 entradas e 20 saídas.

Control Unit / Parallel to Serial

Esta unidade permite passar a informação recebida em paralelo para um formato série. A operação tem de ser executada para diminuir o número de estados necessários na RFSM e o número de entradas nesta. Deste modo, a informação que entra na máquina de estados finitos terá de ser controlada para que não seja enviada informação incorrecta. Assim, este bloco possui igualmente uma função de controlo. Este possui várias entradas que permitem escolher como a informação recebida deve ser tratada. Esta escolha é efectuada pela RFSM.

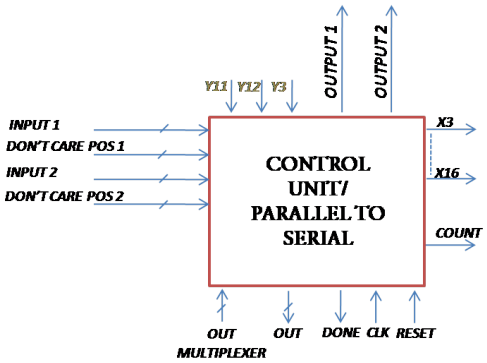

Figura 4.11: Componente *Control Unit / Parallel to Serial*

Observando a Figura 4.11, verifica-se que este possui onze entradas: ***Input 1***, ***Input 2***, ***Don't Care Pos 1***, ***Don't Care Pos 2***, **Y11**, **Y12**, **Y3**, **Y19**, ***Out Multiplexer***, ***Clock*** e ***Reset***:

- ***Input 1*** - Vector de Entrada. Possui um tamanho genérico e dados a analisar;
- ***Input 2*** - Vector de Entrada. Possui um tamanho genérico podendo ter dados ou informações para efectuar operações de comparação;
- ***Don't Care Pos 1*** - Vector de Entrada. Possui um tamanho igual ao vector *Input 1* e indica em que posições deste se encontram os *dont't cares*;
- ***Don't Care Pos 2*** - Vector de Entrada. Possui um tamanho igual ao vector *Input 2* e indica em que posições deste se encontram os *dont't cares*;
- **Y11** - Permite informar este bloco se o vector de entrada 1 e 2 devem ser considerados como vectores binários ou ternários. Caso este sinal se encontre com o valor lógico '0', os vectores de entrada são considerados binários, caso contrário, ternários;
- **Y12** - Uma vez que o valor *don't care* pode ser '1' ou '0' e o simbolo '-' ,o que representa *don't care* em VHDL não é bem interpretado pelo sintetizador da Xilinx, foi considerado um sinal extra que permita escolher em caso de *don't care* (indicado pelo vector Don't Care Pos 1 ou Don't Care Pos 2) se o valor desse vector deve ser considerado como '0' ou como um '1'. Exemplo: Vector1= 1-10 e Y12=0 então o vector que irá entrar na RFSM será 1010. Caso Y12=1 então o vector de saída será 1110.

- **Y3** - Este sinal permite actualizar o valor de saída dos sinais *Output 1* e *Output 2*. Quando o sinal Y3 se encontra a '1', o próximo valor do vector de entrada é colocado nos sinais de saída. Este sinal é controlado pela RFSM.

- **Y19** - Este sinal é muito importante quando em operações entre dois vectores ternários. Este quando possuindo o valor lógico '1' e encontrado um valor *don't care*num dos vectores de entrada o valor de Y12 é colocado não apenas no vector que possui o *don't care*, mas sim em ambos os vectores de entrada. Exemplo: Vector 1: 1-1 Vector 2: 010, Y11='1', Y12='0' e Y19='1' então o vector 1 ficará 101 e o vector 2 000.

- ***Out Multiplexer*** - Este sinal possui o valor de saída da operação e é depois reencaminhado para a saída do processador desenvolvido. Não é ligado directamente à saída do processador, uma vez que este vai alterando ao longo da execução sendo desta forma apenas enviado quando todo o processamento termina.

- ***Clock*** - Sinal de Relógio. Os sinais de saída são actualizados na transição descendente do relógio;

- ***Reset*** - Sinal de *Reset*. Quando activo, os valores de saída são colocados a zero.

Este componente possui 19 saídas: ***Output 1***, ***Output 2***, **X3...X16**, ***Count***, ***Done*** e ***Out***:

- ***Output* 1** - Possui um dos bits do vector de entrada 1. Este valor é actualizado na transição descendente do relógio e quando a entrada Y3 se encontra com o valor lógico '1'.

- ***Output* 2** - Possui um dos bits do vector de entrada 2. Este valor é actualizado na transição descendente do relógio e quando a entrada Y3 se encontra com o valor lógico '1'.

- **X3...X16** - Constituem as restantes entradas para RFSM. São igualmente actualizadas na transição descendente e quando a entrada Y3 se encontra a '1';

- ***Count*** - Esta saída possui o número de bits dos vectores de entrada que já foram enviados para a RFSM. Esta informação será enviada para o bloco *Master Control*;

- ***Done*** – Este toma o valor lógico '1' quando o resultado da operação pretendida se encontra disponivél.

- ***Out*** - Este barramento possui a informação final da execução de uma determinada operação.

Os sinais x3 e x4 são utilizados para indicar à RFSM que uma determinada amostra possui o valor *don't care* ou não. Caso o valor x3 seja '1' então o valor que se encontra na saída *output 1* deverá ser considerado como *don't care*. Caso o valor x4 seja '1' então o valor que se encontra na saída *output 2* deverá ser considerado como *don't care*.

Master Control

Este componente, através da informação recebida pelo bloco *Control Unit*, decide o valor do sinal *Master Enable* utilizado por alguns componentes. Possuí 3 sinais de entrada: *Count*, *Clock* e *Reset*:

- ***Count*** - Informação relativa ao número de elementos dos vectores de entrada que já foram enviados para a RFSM;

- ***Clock*** - Sinal de Relógio. Este componente altera o seu sinal de saída na transição ascendente do relógio;

- ***Reset*** - Sinal de *Reset*.

A saída (**Master Enable**) é actualizada na transição ascendente do relógio e serve para activar/desactivar a actualização de valores de alguns componentes.

4.4.2.3 *Buffer*

O componente *Buffe*r possui uma importância elevada no funcionamento do *datapath*. Observando a Figura 4.10, verifica-se que existem dois blocos que se encontram ligados, possuindo ambos uma actualização dos seus valores nas transições descendentes. Estes são: *Counter/Comparator* e *Max Detector*. Assim, se ambos os componentes fossem ligados directamente, iria causar problemas no bloco *Max Detector*, uma vez que, a sua saída (que depende da entrada) iria estar a ser actualizada ao mesmo tempo que a sua entrada iria ser alterada, levando a um estado de instabilidade. Para resolver este problema, a solução é a utilização de um *buffer* que actualize as suas saídas na transição ascendente do relógio. Assim, o valor de saída do bloco *Counter/Comparator* será actualizado na transição descendente, sendo guardado no buffer na transição ascendente. Na transição descendente seguinte, este valor entra estável no bloco *Max Detector* pelo que a ambiguidade é eliminada.

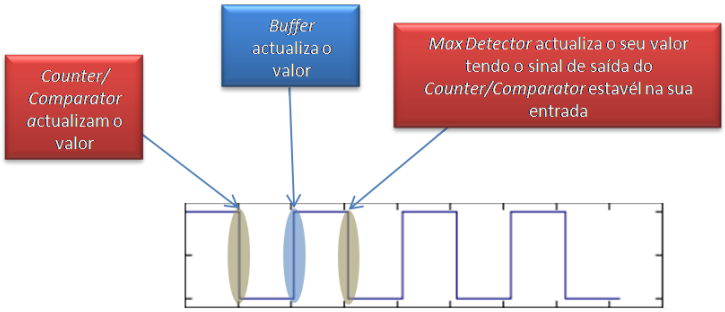

Figura 4.12: Buffer

Esta solução leva a um atraso de um ciclo de relógio até que a operação seja completada. Este facto tem de ser considerado pelo bloco *Master Control* por forma a não desactivar componentes que ainda estejam à espera de informação útil.

4.4.2.4 *Multiplexers* e portas lógicas

Para terminar a explicação da figura 4.10 apenas falta falar sobre os elementos não dependentes do sinal de Clock: Multiplexers e uma porta lógica AND. Os multiplexers são controlados pelas saídas da RFSM, conseguindo-se, desta forma, encaminhar o sinal para os diversos blocos de processamento necessários para a execução da instrução. Um dos Multiplexers permite seleccionar uma das entradas para a RFSM (MUX 2-1) e o outro seleccionar a saída do processador. Em relação às portas lógicas, apenas foi utilizado um AND que permite fazer o 'e' lógico entre as duas primeiras entradas para a RFSM. O resultado desta operação poderá, conforme o valor de Y13, ser escolhido para segunda entrada da RFSM.

4.4.3 Carregamento de Instruções

O carregamento de uma nova instrução é feito através da reprogramação da unidade de controlo, ou seja, da máquina de estados finitos reprogramável. Esta é uma máquina extremamente poderosa, uma vez que consegue desempenhar funções diferentes apenas por alteração de valores lógicos '0' e '1'. No entanto, todos os algoritmos que queremos executar terão de ser bem pensados e enviados de seguida para RFSM. Sendo assim, deve ser possível a comunicação com esta para a sua posterior reprogramação. Ao dispositivo externo que permite esta operação designou-se de **Programador**. A figura seguinte, mostra um macro - esquema da ligação deste ao processador desenvolvido:

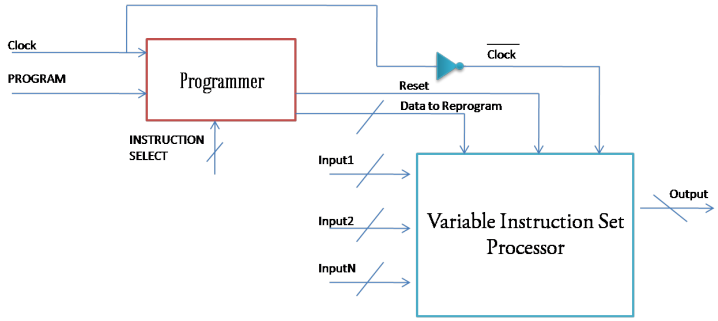

Figura 4.13: Macro - Esquema da ligação do programador ao processador com um conjunto de instruções variável

Através da figura anterior, podemos observar que o programador envia os dados necessários à reprogramação. Este barramento inclui sinais que servem para escrever numa memória específica dentro da RFSM, bem como os dados desta. O sinal de reset da RFSM encontra-se ligado ao programador, uma vez que, no final da fase de programação, a máquina de estados deverá voltar ao estado inicial. O sinal de *Clock* que entra no módulo RFSM encontra-se negado relativamente ao sinal de *Clock* que entra no programador. Isto acontece para que os dados que se encontram no barramento DATA TO REPROGRAM já se encontrem estáveis quando da transição ascendente do clock do módulo RFSM, sendo feita uma correcta reprogramação. Os restantes sinais são explicados a seguir.

Programador

Analisemos agora a estrutura interna do programador. Esta pode ser observada na Figura 4.14.

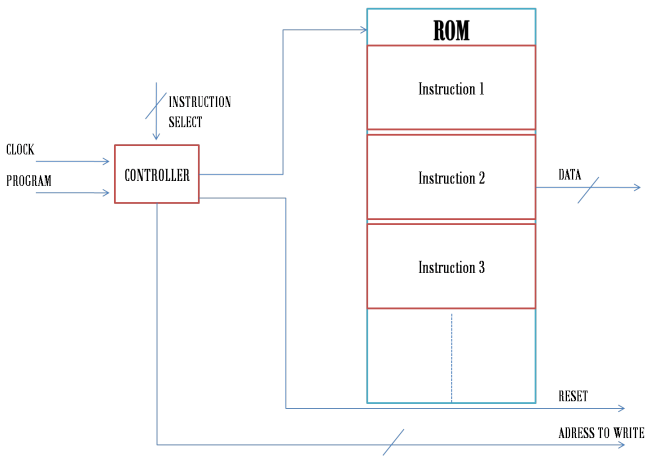

Figura 4.14: Estrutura do programador

Este pode ser dividido em dois sub-componentes: *Controller* e Memória ROM.

Através do componente *Controller*, é possível endereçar a posição de memória (Memória ROM) onde se encontra a informação sobre a instrução pretendida, sendo esta carregada de seguida para a memória respectiva dentro da unidade de controlo.

A selecção da instrução é feita através do barramento *INSTRUCTION SELECT*. O sinal de *PROGRAM* serve para fazer o pedido de reprogramação da unidade de controlo. Quando o processo de reprogramação estiver terminado, o sinal *RESET* é activado, colocando, deste modo, a RFSM no seu estado inicial. Os sinais *DATA* e *ADRESS TO WRITE* encontram-se na Figura 4.13 representados pelo barramento *DATA TO REPROGRAM*. A informação na memória (Memória ROM) encontra-se estruturada para que apenas uma das memórias da RFSM se encontre a ser escrita num determinado instante de tempo. O seu formato encontra-se representado na figura seguinte:

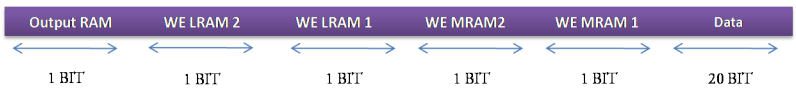

Figura 4.15: Trama de reprogramação

Observando a figura anterior, verifica-se que existe 20 bits de dados. Este valor tem de coincidir com o tamanho máximo das palavras encontrado nas memórias da RFSM. No nosso caso, a memória com a palavra de tamanho máximo é a memória de saída (Output RAM) com 20 saídas, pelo que o número de bits do campo Data terá de possui 20 bits. De seguida, encontram-se os bits que activam a memória para o qual se vai escrever a informação do campo Data. Vamos observar um exemplo para uma melhor compreensão: Considere que o barramento *ADRESS TO WRITE* se encontra com o valor 0 e que o componente *CONTROLLER* se encontra a endereçar o seguinte conteúdo de memória.

Figura 4.16: Exemplo de uma trama de reprogramação

Uma vez que o último bit se encontra com o valor lógico '1', a escrita na memória de saída será activada. Assim, os dados 00000000000000001 serão escritos no endereço 0 da memória de saída.

4.4.4 Fluxograma reprogramável

Este bloco tem como objectivo através de uma determinada entrada seleccionar a instrução de que deverá ser executada. Este vai permitir uma maior abstracção de camadas superior. Para observar este facto, vamos considerar o seguinte fluxograma:

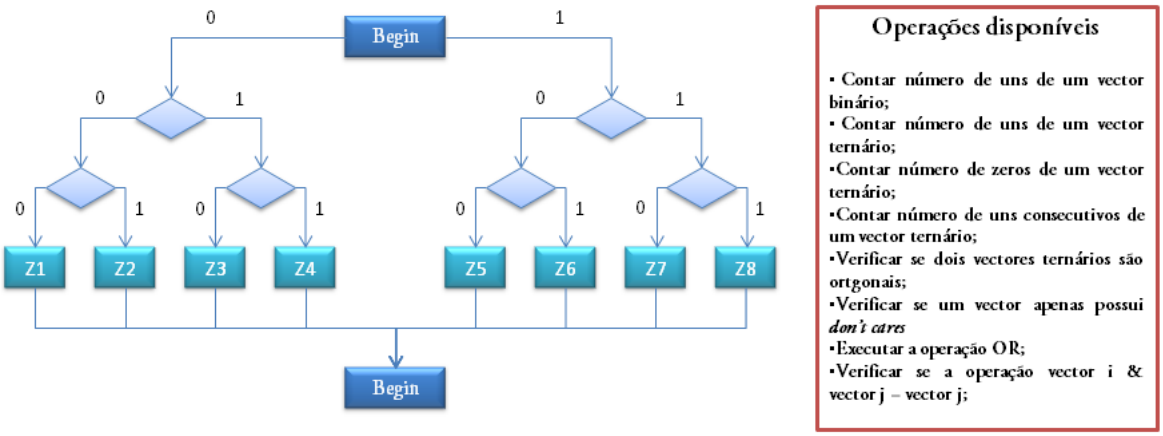

Figura 4.17: Exemplo de Fluxograma

No exemplo apresentado, representaram-se as instruções pelas siglas Z1 até Z8.

Vamos considerar, como exemplo, que temos ao dispor as operações apresentadas na caixa a vermelho. Depois da reprogramação, vamos supor que a correspondência de cada uma das operações é a seguinte:

Operação	Correspondência
Contar o número de uns de um vector binário	Z1
Contar o número de uns de um vector ternário	Z2
Contar o número de zeros de um vector ternário	Z3
Contar o número de uns consecutivos de um vector ternário	Z4
Verificar se dois vectores ternários ortogonais	Z5
Verificar se um vector apenas possui valores don't care	Z6
Executar a operação OR	Z7
Verificar se a operação vector i & vector j = vector j	Z8

Tabela 4.1: Correspondência Operação - Módulo

Observando a figura, vemos que se o vector de entrada for 000, a instrução Z1 será escolhida sendo, desta forma, executada a operação Contar número de uns de um vector binário. Este vector de entrada, encontra-se representado na Figura 4.8 como sendo o sinal In_Vect. Imaginemos agora que, devido a uma posterior reprogramação, a correspondência passou a ser a seguinte:

Operação	Correspondência
Contar o número de uns de um vector binário	Z2
Contar o número de uns de um vector ternário	Z1
Contar o número de zeros de um vector ternário	Z4
Contar o número de uns consecutivos de um vector ternário	Z3
Verificar se dois vectores ternários ortogonais	Z6
Verificar se um vector apenas possui valores don't care	Z5
Executar a operação OR	Z8
Verificar se a operação vector i & vector j = vector j	Z7

Tabela 4.2: Correspondência Operação - Módulo após reprogramação

Deste modo, segundo o fluxograma anterior, para executar a operação Contar número de uns de um vector binário, seria necessário introduzir como vector de entrada 001. Deste modo, exigia-se que a entidade que chama o processador soubesse à priori que a correspondência teria mudado. Assim, para que a haja uma abstracção completa de camadas superior, verifica-se a necessidade da reprogramação do fluxograma. Através desta, apesar da mudança da correspondência, podemos reconfigurar o fluxograma para que o módulo a ser chamado quando o vector de entrada In_Vect for 000 continue a ser contar número de uns de um vector binário.

Esta é a principal razão para a construção desta entidade reprogramável. Procedamos, de seguida, à explicação da construção deste módulo.

Implementação

A implementação desta entidade é em tudo semelhante à unidade de controlo do processador com um conjunto de instruções variável. Neste caso, apenas é necessário calcular o módulo que deverá ser activado num determinado instante de tempo. Este cálculo pode ser feito directamente sobre os estados da máquina de estados finitos reprogramável e, consoante o estado final desta, o valor da memória de saída irá conter a informação sobre qual a instrução a executar. Um exemplo encontra-se na Figura 4.18.

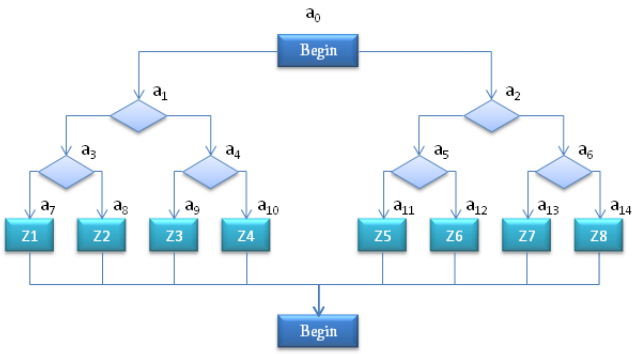

Figura 4.18: Cálculo da instrução a executar

Observando a figura anterior verifica-se que neste exemplo se o estado final obtido coincidir com a7, a instrução que irá ser executada será a indicada por Z1. Assim no endereço 7 da memória de saída o conteúdo terá de ser 0001 (Caso a memória seja de 4 bits).

Existem duas implementações possíveis para elaborar esta entidade. Ambas consistem no modelo em cascata sendo que a única diferença é a utilização ou não de *Dummy States*. Existem vantagens e desvantagens de ambas as implementações:

RFSM s/ Dummy States

Vantagens

- Exige um menor número de memórias RAM e multiplexers;
- Módulo de reprogramação mais simples;
- Trama de reprogramação pequena;
- Menor informação necessária para reprogramação;

Desvantagens

- Maior tempo de processamento;
- Num ciclo de relógio apenas é possível a análise de uma entrada;

RFSM c/ Dummy States

Vantagens

- Cálculo do estado final possível em apenas um ciclo de relógio (3 andares para uma entrada In_Vect de 3 bits);
- Possível a análise de várias entradas em apenas um ciclo de relógio;

Desvantagens

- Trama de reprogramação maior relativamente ao caso anterior;
- Módulo de reprogramação mais complexo;
- Maior informação necessária para reprogramação;

Tendo em conta que as dimensões desta máquina de estados são relativamente pequenas, a dificuldade de todos os componentes envolvidos na reprogramação não terão uma grande complexidade. Assim, analisando as vantagens e desvantagens de ambas as implementações, verificamos que as vantagens da primeira opção (S/ Dummy States) apenas se restringem à menor complexidade dos elementos em causa. Deste modo, como a complexidade do sistema não é elevada, optou-se pela utilização da segunda implementação. Esta permite uma maior rapidez no cálculo do módulo que deverá ser escolhido, uma vez que se utilizarmos uma cascata com o mesmo número de entradas, conseguimos obter o resultado final em apenas um ciclo de relógio. Foi utilizada uma cascata de três andares, permitindo assim escolher entre 8 módulos, tal como representado na Figura 4.18.

Reprogramação do fluxograma

Como no caso do bloco *Variable Instruction Set Processor*, terá de existir uma entidade que efectue a reprogramação da máquina de estados finitos reprogramável. A ligação deste módulo com o fluxograma reprogramável é em tudo semelhante ao apresentado na Figura 4.13. Um esquema deste encontra-se representado na figura seguinte.

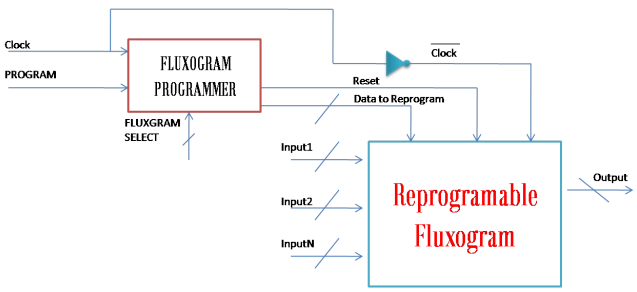

Figura 4.19: Macro - Esquema da reprogramação do fluxograma

Analisando o esquema da Figura 4.19, verifica-se a sua semelhança com o da Figura 4.13. A única diferença será nas tramas de reprogramação, uma vez que serão utilizados 3 andares na máquina de estados reprogramável e a entrada *Instruction Select*, que passa a designar-se *Fluxogram Select*. Esta entrada irá permitir escolher um dos fluxogramas que se encontra retidos em memória. Será, de seguida, analisado o bloco Fluxogram Programmer.

Fluxogram Programmer

A estrutura do programador é em tudo idêntica ao programador do bloco *Variable Instruction Set Processor*. A Figura 4.20 apresenta esta estrutura:

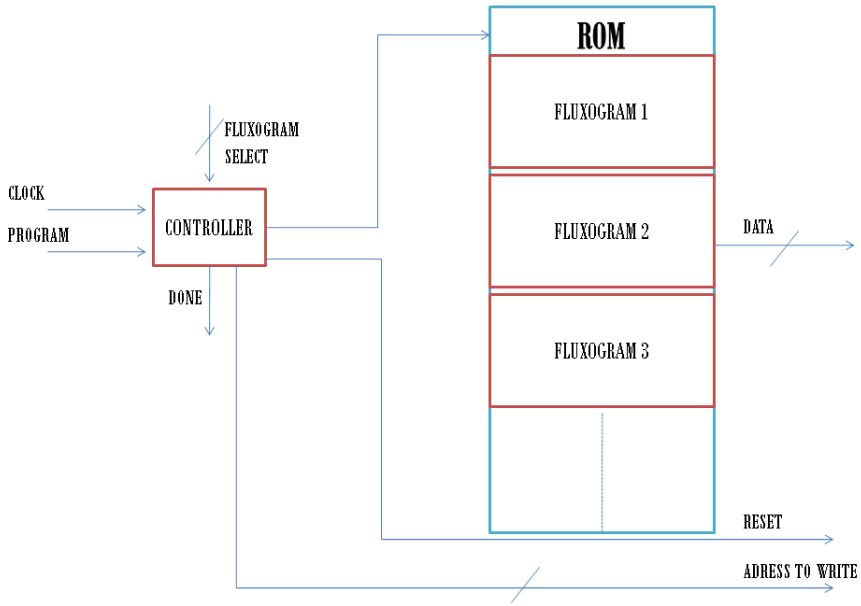

Figura 4.20: Programador do Fluxograma

Observando a figura anterior, verifica-se imediatamente a sua semelhança com a Figura 4.14. No entanto, nesta verifica-se a existência de um sinal de Done. A sua existência irá permitir saber quando a operação do cálculo do módulo a executar se encontra completa. Este sinal pode ser implementado directamente no programador, uma vez que sabemos à partida quantos ciclos de relógio demora o cálculo da instrução a executar. No nosso caso, este demora apenas um ciclo de relógio após a reprogramação da máquina de estados finitos. Vamos agora analisar a estrutura da trama de reprogramação do fluxograma. Esta encontra-se representada na figura seguinte:

Figura 4.21: Trama de reprogramação do fluxograma

Observando esta, facilmente se verifica a sua semelhança com trama de reprogramação da unidade de controlo do processador desenvolvido. Esta, no entanto, possui mais dois campos: WE LRAM3 e WE MRAM 3. Estes dois campos dizem respeito ao terceiro andar da máquina de estados reprogramável. A segunda diferença em relação à figura Figura 4.15 é o número de bits do campo Data. Esta contém apenas 4 Bits, sendo o suficiente se observarmos a Figura 4.17.

4.4.5 Data Control Unit

Depois de uma compreensão de como funciona o processador com um conjunto de instruções variável e como é possível escolher a instrução pretendida através de um fluxograma reprogramável, estamos em condições de interligar os blocos apresentados até este momento. Para tal, terá de existir uma entidade superior a todos estes blocos que controle o fluxo de informação entre eles, bem como a sincronização de operações (ex: uma instrução não poderá iniciar sem que o cálculo efectuado pelo fluxograma reprogramável se encontre terminado). A esta unidade designaremos de *Data Control Unit* e encontra-se representada no macro esquema da Figura4.8.

Podemos então sintetizar as funções que esta unidade deverá desempenhar:

- Transferência de informação de entrada para os componentes de nível mais baixo;
- Inicialização do cálculo da instrução que deverá ser executada;
- Transferência dos resultados do bloco Reprogramable Fluxogram para o bloco *Variable Instruction Set Processor*;
- Controlo dos dados de saída resultantes da execução de uma instrução;

Depois de definido as funções desta unidade de controlo, é possível a construção de um diagrama de fluxo de sinal que nos permitira elaborar de uma forma mais simples um código para uma linguagem de programação. Este diagrama encontra-se representado na figura seguinte:

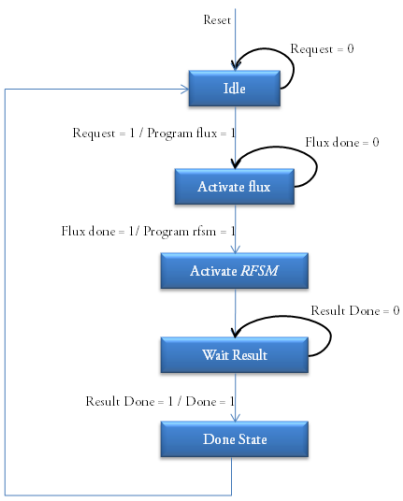

Figura 4.22: Diagrama de Fluxo de Sinal da Unidade de Controlo Central

Para a implementação desta unidade de controlo utilizou-se uma máquina de estados finitos simples, não possuindo características reconfiguráveis, uma vez que o seu diagrama é constante para qualquer que seja a operação. Vamos proceder então à explicação do diagrama anterior:

Esta máquina permanece no estado *Idle* (Desocupado) até que seja feito um pedido exterior. Este pedido é feito através do sinal Request. Após este, haverá uma transição de estado. Nesta transição, o sinal *Program flux* tomará o valor '1'. Este sinal liga à entrada de *Program* do programador do fluxograma. Assim, este sinal irá desencadear a reprogramação da máquina de estados finitos relativa ao fluxograma.

Depois de a reprogramação ter terminado, é feita a execução do algoritmo no fluxograma. A unidade de controlo fica então à espera que o sinal *Flux Done*, vindo do programador do fluxograma, tome o valor lógico '1'. Este indica que o cálculo da instrução a executar se encontra terminado, sendo já possível a execução da mesma.

De seguida a máquina de estados de controlo transita para o estado *Activate RFSM*. Durante a transição de estado, o sinal *PROGRAM RFSM* que se encontra ligado ao programador da unidade de controlo passa para o valor lógico '1'. Esta mudança irá desencadear uma reprogramação da unidade de controlo do processador sendo actualizada com a informação da instrução pretendida. Após esta, a máquina de estados transita para o estado *Wait Result*, ficando neste à espera que o resultado da instrução termine. Quando este terminar, o sinal *Result Done* irá tomar o valor lógico '1'. Este indica que o resultado da operação pretendida já se encontra disponível.

Para terminar, a unidade de controlo transita para o estado *Done State*, em que nesta transição um sinal *Done* fica com o valor lógico '1', indicando assim às camadas superiores que o resultado da operação pretendida já se encontra disponível. Estando neste momento todas as operações terminadas, a unidade de controlo transita novamente para o estado *Idle*, ficando neste até que um novo pedido seja efectuado.

4.4.6 *Instruction Set Desenvolvido*

Como referido anteriormente, visto o datapath ser fixo, este permite a execução de um número limitado de instruções, podendo estas ser carregadas conforme a necessidade. As seguintes instruções foram desenvolvidas:

- Contar o número de uns de um vector binário;
- Contar o número de uns de um vector ternário;
- Contar o número de zeros de um vector ternário;

- Contar o número máximo consecutivo de uns de um vector binário;
- Contar o número máximo consecutivo de uns de um vector ternário;
- Contar o número máximo consecutivo de zeros de um vector ternário;
- Verificar se um vector ternário apenas contém valores *dont care*;
- Verificar se um vector contém apenas uns/zeros;
- Verificar se um vector não contém apenas uns/zeros;
- Verificar se um vector contém K elementos a um/zero;
- Verificar a expressão *vectori and vectorj = vectorj;*
- Executar a operação vectori or vectorj;
- Verificar se dois vectores ternários são ortogonais;

4.5 Processador Desenvolvido para Co-processamento

O processador desenvolvido não permite a execução de um programa sequencial ao qual estamos mais frequentemente habituados. Este serve, deste modo, para efectuar co-processamento, isto é, é útil para uma ligação a um processador de uso geral, sendo, a partir deste, chamado para efectuar operações reduzindo a carga no processador principal. Para demonstrar o funcionamento do processador desenvolvido, este foi acoplado a um processador de uso geral simples desenvolvido na universidade de Rostock. [53]

Para a demonstração do processador desenvolvido, foram elaborados dois algoritmos de pesquisa combinatória: **cobertura de matriz** e **satisfação booleana**. Os algoritmos optimizados para estes problemas podem ser encontrados em [29] e [54].

Estrutura do Sistema Computacional de uso geral utilizado

A estrutura do sistema base utilizado encontra-se representada na figura seguinte:

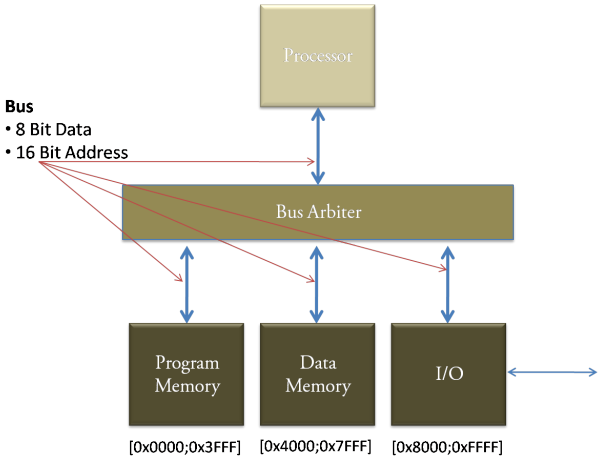

Figura 4.23: Estrutura do processador base de uso geral utilizado

Este sistema apresenta 5 componentes base: **Processador**, **"Arbitro" do barramento**, **memória para o programa**, **memória para dados** e um **bloco I/O.**

Observando a figura 4.23, verifica-se a existência de um barramento partilhado no qual se ligam a memória que contém o programa, memória para dados e bloco de I/O, sendo o acesso ao barramento gerido por um "Árbitro". Este barramento possui 8 bits de dados e 16 para endereço. O acesso aos diferentes componentes encontra-se mapeado, sendo a gama [0x0000;0x3FFF] reservada à memória do programa, [0x4000;0x7FFF] para a memória de dados e [0x8000;0xFFFF] para dispositivos I/O. Visto existir mais do que um dispositivo I/O, esta gama encontra-se subdividida em várias sub-gamas conforme o dispositivo requerido.

Processador Base

O processador base possuí uma estrutura muito simples. Este incorpora um registo de uso geral (acumulador), o qual serve para efectuar cálculos intermédios, um *instruction set* com instruções de salto condicional, salto incondicional e instruções aritméticas. Este encontra-se em anexo.

O fluxo de operações do processador é muito simples. Possui 4 estados de execução: **Fetch Op**, **Fetch Addr High**, **Fetch Addr Low** e **Execute**.

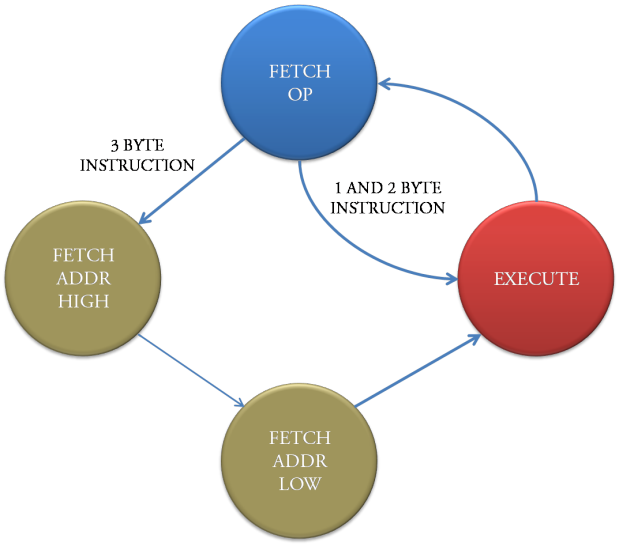

Figura 4.24: Diagrama de execução do processador

O processador começa com o habitual *fetch*, no qual é identificada a operação a executar através do seu opcode. Neste, existem instruções com tamanho entre 1 e 3 *bytes*. As instruções com 1 ou 2 *bytes* apenas possuem dois estados de execução: fetch op e execute, sendo o segundo *byte* adquirido no estado execute. Nas instruções de 3 *bytes,* o fluxo de execução passa por dois estados adicionais: Fetch Addr High e Fetch Addr Low, nos quais são adquiridos os valores dos 2^{o} e 3^{o} *bytes*, passando depois para a fase de execução.

Alteração do Sistema Computacional de uso geral

De forma a conseguir executar os algoritmos propostos para demonstração ao sistema de uso geral apresentado, foram feitas algumas alterações. Visto o número de registos base ser muito reduzido, começou-se pela adição de novos registos (8 bits) de uso geral (16 ao todo). De seguida, foi adicionado uma stack, permitindo, desta forma, guardar os valores dos registos, de modo a encontrar a melhor solução para o problema combinatório. Tendo como objectivo a execução de algoritmos sobre matrizes binárias e ternárias. Foram adicionadas duas memórias extra, contendo uma delas, os valores da matriz e a segunda indicando quais destes valores devem ser considerados como *don't care*. Por fim, foi adicionado um registo que possuí o valor da solução num determinado instante.

De forma a interagir com estes novos elementos, foram adicionadas novas instruções ao processador base (ver anexo), sendo igualmente adicionada um novo estado do fluxo de execução permitindo instruções até 4 bytes: **get_last_byte**.

Ligação ao processador combinatório desenvolvido

Depois da adição de novos elementos ao processador, foi feita a ligação ao processador combinatório desenvolvido. Existem várias formas de ligação deste ao processador de uso geral (designaremos a partir deste ponto o processador combinatório por co-processador):

- **Por mapeamento**, isto é, ligar o co-processador ao barramento de dados/endereços existente.
- **Ligação directa**, isto é, criar uma ligação directa, sem mais nenhum interveniente entre o processador e o co-processador;
- **Ligação por I/O**, isto é, ligar o co-processador ao bloco I/O e considerando-o como um dispositivo I/O.

Foi utilizada uma ligação directa para conexão entre os dois componentes, permitindo desta forma uma interface mais simples entre processadores.

A figura seguinte mostra a ligação feita, bem como os novos componentes incorporados no processador de uso geral.

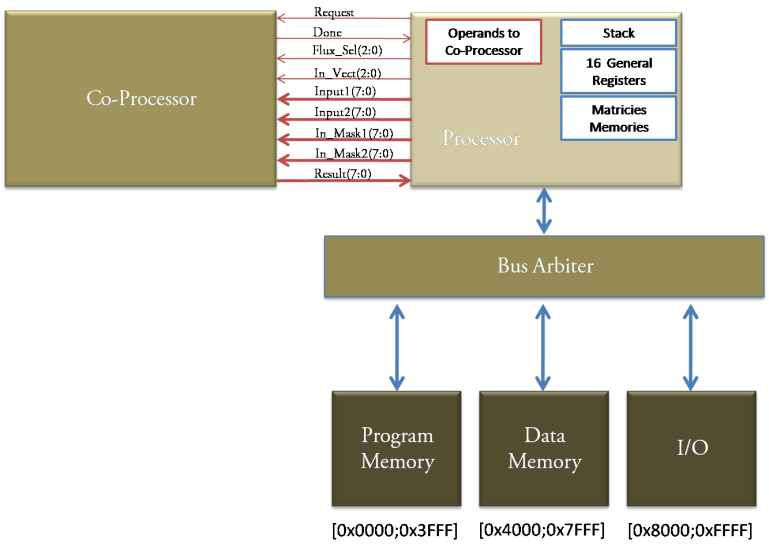

Figura 4.25: Ligação ao processador programável e novos componentes do processador de uso geral

Para além dos componentes adicionados acima referidos, foram incorporados no processador mais 7 registos que se encontram designados na figura anterior como *Operands to Co-Processor*. Estes registos são escritos através de instruções próprias e

servem para indicar quando na chamada ao co-processador qual o fluxograma, operação e operandos (vectores binários e ternários) que irão ser analisados, sendo estes valores automaticamente direccionados para as entradas do co-processador. Quando da escrita de todos os registos necessários, é necessário pedir ao co-processador para que processe a informação. Isso é feito pela instrucção ***reconfigInstr,REG***, em que *Reg* é o registo para o qual o resultado da operação deverá ser guardado. Quando carregada esta instrução da memória do programa, o sinal *request* para o co-processador é activado. Quando feito este pedido, o processador fica à espera que o sinal *Busy* tome o valor lógico 1. Quando tal acontece, o valor indicado pelo sinal *result* é lido e guardado num dos 7 registos do bloco *Operands to Co-Processor,* bem como no registo indicado pela instrução (*REG*). Assim, o processador bloqueia quando existe um pedido ao co-processador, estando, deste modo, a gastar recursos desnecessariamente. No entanto, não é objectivo deste trabalho a optimização mas sim demonstração, pelo que será mantida esta política.

4.6 Resultados e Discussão

Para a demonstração do processador desenvolvido, este foi interligado a um processador de uso geral, o qual foi modificado de forma a permitir a execução de algoritmos de pesquisa combinatória. De seguida, são apresentados alguns resultados obtidos, principalmente a nível de ocupação de recursos dos diversos componentes.

Foram desenvolvidos, para demonstração, os seguintes algoritmos de pesquisa combinatória: **satisfação booleana** e **cobertura de matriz**. De forma, para uma melhor percepção do sistema algumas informações foram apresentadas no ecrã VGA. Toda a interface com o monitor VGA e teclado pode ser encontrada em [55].

São apresentados agora alguns resultados no que diz respeito à ocupação da FPGA a nível de recursos.

Unidade de controlo reprogramável (RFSM)

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
RFSM	3/49	4/4	0/80	0/76	0/0	0/0	0/0	0/0
[-] FSM_CC[1].Level_PM	0/16	0/0	0/26	0/24	0/0	0/0	0/0	0/0
[-] Level_MUX	0/8	0/0	0/10	0/8	0/0	0/0	0/0	0/0
MUX	4/4	0/0	2/2	0/0	0/0	0/0	0/0	0/0
MUX_RAM	4/4	0/0	8/8	8/8	0/0	0/0	0/0	0/0
Level_RAM	8/8	0/0	16/16	16/16	0/0	0/0	0/0	0/0
[-] FSM_CC[2].Level_PM	0/16	0/0	0/26	0/24	0/0	0/0	0/0	0/0
[-] Level_MUX	0/8	0/0	0/10	0/8	0/0	0/0	0/0	0/0
MUX	4/4	0/0	2/2	0/0	0/0	0/0	0/0	0/0
MUX_RAM	4/4	0/0	8/8	8/8	0/0	0/0	0/0	0/0
Level_RAM	8/8	0/0	16/16	16/16	0/0	0/0	0/0	0/0
Output_RAM	14/14	0/0	28/28	28/28	0/0	0/0	0/0	0/0

Tabela 4.3: Utilização de recursos da unidade de controlo reprogramável

A tabela 4.5, apresenta um resumo dos recursos no que diz respeito à unidade de controlo do processador com um *instruction set* variável, construída a partir de uma máquina de estados finitos reprogramável. Observando os resultados anteriores, é possível confirmar as expectativas previstas no que diz respeito à ocupação de recursos da unidade de controlo, verificando-se que estes são muito reduzidos, levando a uma grande motivação para o seu uso. Foi, assim, confirmado uma das grandes vantagens da utilização de um processador com um *set de instruções* variáveis. É de notar igualmente que este trabalho não teve como objectivo a optimização dos componentes, pelo que este pode ainda ser melhorado, diminuindo ainda mais os recursos ocupados. É de salientar que estes dependem do número de entradas, saídas, estados e da quantidade de andares que compõe a RFSM. No entanto, os valores destes parâmetros utilizados já são significativos.

Fluxograma reprogramável (Reprogrammable Fluxogram)

A tabela seguinte mostra a utilização de recursos do componente *reconfigurable Fluxogram*:

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
Reprogrammable Fluxogram	3/45	4/4	0/78	0/72	0/0	0/0	0/0	0/0
[-] FSM_CC[1].Level_PM	0/13	0/0	0/24	0/22	0/0	0/0	0/0	0/0
[-] FLUX_Level_MUX	0/5	0/0	0/8	0/6	0/0	0/0	0/0	0/0
MUX	2/2	0/0	2/2	0/0	0/0	0/0	0/0	0/0
MUX_RAM	3/3	0/0	6/6	6/6	0/0	0/0	0/0	0/0
FLUX_Level_RAM	8/8	0/0	16/16	16/16	0/0	0/0	0/0	0/0
[-] FSM_CC[2].Level_PM	0/13	0/0	0/24	0/22	0/0	0/0	0/0	0/0
[-] FLUX_Level_MUX	0/5	0/0	0/8	0/6	0/0	0/0	0/0	0/0
MUX	2/2	0/0	2/2	0/0	0/0	0/0	0/0	0/0
MUX_RAM	3/3	0/0	6/6	6/6	0/0	0/0	0/0	0/0
FLUX_Level_RAM	8/8	0/0	16/16	16/16	0/0	0/0	0/0	0/0
[-] FSM_CC[3].Level_PM	0/13	0/0	0/24	0/22	0/0	0/0	0/0	0/0
[-] FLUX_Level_MUX	0/5	0/0	0/8	0/6	0/0	0/0	0/0	0/0
MUX	2/2	0/0	2/2	0/0	0/0	0/0	0/0	0/0
MUX_RAM	3/3	0/0	6/6	6/6	0/0	0/0	0/0	0/0
FLUX_Level_RAM	8/8	0/0	16/16	16/16	0/0	0/0	0/0	0/0
Output_RAM	3/3	0/0	6/6	6/6	0/0	0/0	0/0	0/0

Tabela 4.4: Utilização de recursos do fluxograma reprogramável

Tal como foi referido anteriormente, este componente é igualmente baseado em máquinas de estados finitos reconfiguráveis. Observando a tabela apresentada, é possível verificar a semelhança de resultados com a anterior. É de salientar que este possuí um número de estados, saídas e entradas menores que o componente anterior mas, devido ao maior número de andares, possuí uma ocupação comparativamente ao anterior semelhante.

Processador com instruction set variável

A tabela seguinte mostra a ocupação de recursos utilizada pelo processador construído:

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
Variable Inst. Set Processor	616/2913	21/263	1206/5547	0/3988	0/0	0/0	0/0	0/0
[-] FLUXOGRAM	4/816	0/70	8/1560	0/1480	0/0	0/0	0/0	0/0
FLUX	3/45	4/4	0/78	0/72	0/0	0/0	0/0	0/0
FLUX_ROM	704/704	0/0	1408/1408	1408/1408	0/0	0/0	0/0	0/0
Flux_Controller	63/63	66/66	66/66	0/0	0/0	0/0	0/0	0/0
[-] Processor	9/1481	0/172	17/2781	0/2508	0/0	0/0	0/0	0/0
RFSM_Controller	69/69	64/64	79/79	0/0	0/0	0/0	0/0	0/0
Datapath_Module	130/138	104/104	157/173	0/0	0/0	0/0	0/0	0/0
RFSM	3/49	4/4	0/80	0/76	0/0	0/0	0/0	0/0
ROM	1216/1216	0/0	2432/2432	2432/2432	0/0	0/0	0/0	0/0

Tabela 4.5: Utilização de recursos do processador com instruction set variável

A tabela 4.5, apresenta os resultados dos diversos componentes que constituem o processador desenvolvido. Este divide-se em dois componentes principais: Fluxogram

e Processor, vistos já anteriormente. Em relação ao primeiro, é possível verificar que o seu tamanho depende muito do programador (FLUX_ROM + Flux_Controller) mais especialmente da memória ROM que contêm as informações sobre os diversos fluxogramas. O tamanho ocupado por esta memória, deve-se ao facto de ser implementado recorrendo a memória distribuída, isto é, a memória encontra-se implementada em blocos CLB. Uma forma de reduzir este tamanho, consiste na utilização de block RAMs o que se pretende fazer em projectos futuros. Em relação ao componente Flux_Controller este possuí um tamanho reduzido.

Em relação ao componente Processor, as conclusões são em tudo idênticas ao componente anterior. No entanto, é de notar que o *datapath* construído ocupa relativamente pouco, sendo, no entanto, possível uma futura optimização bem como da memória ROM que possuí as informações do *instruction set* necessário.

Sistema constituido pelos dois processadores (combinatório e uso geral) bem como restantes componentes

A tabela seguinte, apresenta a utilização de recursos considerando o sistema representado na figura 4.25.

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
Inst_system	25/8012	34/3011	23/13099	0/3988	0/2	0/0	1/2	0/0
Arb_i	13/13	0/0	22/22	0/0	0/0	0/0	0/0	0/0
Io_i	11/11	16/16	12/12	0/0	0/0	0/0	0/0	0/0
Proc_i	5050/5050	2698/2698	7495/7495	0/0	0/0	0/0	1/1	0/0
Ram_i	0/0	0/0	0/0	0/0	1/1	0/0	0/0	0/0
Reconf_prc	616/2913	21/263	1206/5547	0/3988	0/0	0/0	0/0	0/0
Prog Mem	0/0	0/0	0/0	0/0	1/1	0/0	0/0	0/0

Tabela 4.6: Utilização de recursos do sistema constituido pelos dois processadores (combinatório e uso geral) bem como restantes componentes

Observando a tabela é possível verificar tal como seria de esperar os componentes que ocupam mais espaço na FPGA são os processadores. No entanto estes podem ser bastante optimizados. Em relação ao processador com um *instruction set variavel* já foram apresentadas as razões da ocupação desnecessária do espaço. Em relação ao processador de uso geral (Proc_i) este usa extensivamente variáveis, bem como a utilização de ciclos *for*, provocando inevitavelmente um aumento brusco dos recursos utilizados.

Este sistemas foi implementado na placa de desenvolvimento Celoxica RC10, o qual incorpora uma FPGA Spartan 3 XC3S1500L. Por forma a uma melhor percepção dos

recursos utilizados a tabela seguinte mostra em percentagem a quantidade de recursos desta FPGA utilizado pelos principais componentes que compõem o sistema desenvolvido:

Module	Slices	Slice Reg	LUTs
General Purpose Processor	≈ 38%	≈ 10%	≈ 28%
Variable Inst. Set Processor			
[-] FLUXOGRAM			
FLUX	≈ 0.3%	≈ 0.01%	≈ 0.3%
FLUX_ROM	≈ 5%	≈ 0%	≈ 5%
Flux_Controller	≈ 0.5%	≈ 0.2%	≈ 0.2%
[-] Processor			
RFSM_Controller	≈ 0.5%	≈ 0.2%	≈ 0.3%
Datapath_Module	≈ 1%	≈ 0.4%	≈ 0.6%
RFSM	≈ 0.3%	≈ 0.02%	≈ 0.3%
ROM	≈ 9%	≈ 0%	≈ 9%

Tabela 4.7: Utilização em percentagem dos recursos dos principais componentes

Frequência Máxima de operação

Através das recentes ferramentas de síntese, é possível hoje saber a frequência máxima de operação que o sistema projectado pode funcionar. Para o sistema em causa, obteve-se uma frequência máxima de 16MHz, o que levou à necessidade da redução da velocidade do relógio. Estas ferramentas, permitem igualmente saber qual o caminho do sistema desde o nível mais alto da hierarquia do projecto que limita a frequência máxima de trabalho. Foi observado que, o caminho envolvia o processador de uso geral. As razões para esta redução são as mesmas apresentadas para o caso da utilização de recursos enunciadas anteriormente. Por forma a verificar o desempenho do processador construído, o processador de uso geral foi retirado, bem como todos os componentes acoplados a este e interface gráfica. Depois deste procedimento, a ferramenta de síntese devolveu um valor de aproximadamente 110MHz, ou seja, aproximadamente uma ordem de grandeza superior, indicando mais uma vez que o processador de uso geral necessita de uma grande optimização. Este valor no entanto não reflecte a velocidade do processador com um *instruction set variável*, isto é, uma vez que é necessário reprogramar a unidade de controlo, existe uma grande latência entre o pedido para a execução de uma instrução e a própria execução, constituindo deste modo a principal desvantagem da utilização de uma unidade de controlo reprogramável. Para reduzir esta latência, é necessário que a escrita nas memórias que constituem a unidade de controlo seja mais eficiente, constituindo deste modo um tema para projectos futuros.

Capítulo 5

Interacção Remota

5.1 Sumário

Neste capítulo será apresentado todo o trabalho realizado em termos de interacção remota.

Este começa com a apresentação do módulo CC1101 da *Texas Instruments*, características mais relevantes e a ligação deste às placas de desenvolvimento. De seguida, são apresentados os pontos mais relevantes que têm de ser contabilizados para o desenvolvimento da *interface* entre o módulo CC1101 e FPGAs.

Para finalizar, é apresentada a interface construída para comunicação remota, ligação desta com o restante sistema computacional e resultados práticos obtidos.

5.2 Introdução

Tendo este trabalho como um dos objectivos a interacção remota entre FPGAs, existe a necessidade da utilização de um módulo *wireless.*

Objectivos

Para a escolha do módulo wireless é necessário que este consiga atingir os seguintes objectivos:

- **Possibilidade de alteração do *set* de instruções remotamente, bem como o carregamento de um novo programa *assembly.***
- **Possibilidade futura da criação de uma rede de processadores**, permitindo processamento paralelo;

- **Baixo Custo**;
- **Distância Razoável**;
- **Taxas de transmissão no mínimo da ordem dos Kbits/s**;
- **Ligação Bidireccional**;

O *Transceiver* escolhido foi o CC1101 da Texas Instruments. As suas aplicações comerciais encontram-se na área de sistemas de baixa potência, monitorização industrial e de controlo, automação, redes *wireless* de sistemas de segurança e redes de sensores;

Características do *Transceiver* CC1101

As características mais relevantes são apresentadas de seguida.

Características de desempenho em rádio frequência

- Sensibilidade que pode chegar aos -112dBm;
- Corrente de consumo no máximo de 32mA;
- Potência de saída até +10dBm;
- Taxa de transmissão programável de 1.2 até 500Kbaud;
- Frequências de trabalho: 300-348MHz, 387-464MHz e 779-928MHz;

Características analógicas

- Vários tipos de modulaçoes:2-FSK, GFSK, MSK, OOK e ASK;
- *Frequency Hopping*;
- Compensação automática da frequência central para um perfeito alinhamento com a portadora recebida;
- Sensor de temperatura integrado;

Características digitais

- Envio de palavras de sincronização, detecção de endereços e correctores de erro feitos automaticamente;
- Interface SPI a 4 fios, com possibilidade de modo *burst*;
- Saída digital do indicador RSSI;

- Largura de banda do filtro de canal programável;
- Indicador digital do indicador CCA;
- FIFO de recepção e transmissão separadas com tamanho de 64Bytes;

Outras características

- Funcionalidade *Wake On Radio*;
- O consumo máximo no estado *Sleep* é 400nA;
- Tempo de passagem no máximo de 250us do estado *Sleep* para o estado de recepção ou transmissão;

Descrição circuito

Um diagrama simplificado do circuito integrado encontra-se na figura 5.1.

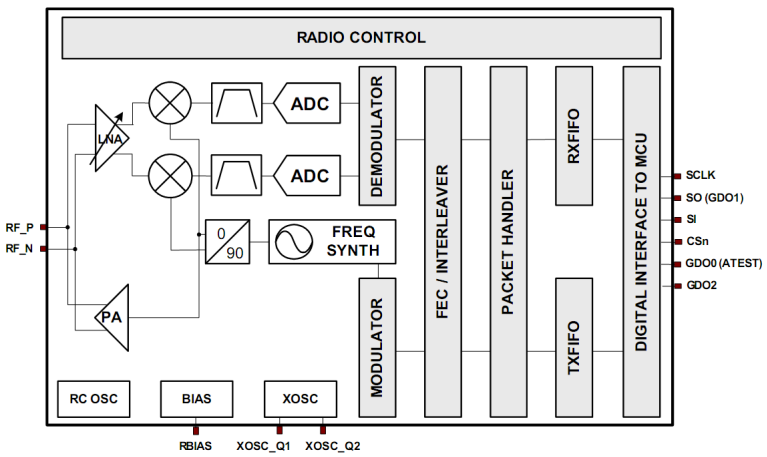

Figura 5.1: Diagrama de blocos simplificado do circuito integrado CC1101 [24]

Sendo este componente um *transceiver*, é possível distinguir os ramos de recepção e de transmissão.

Recepção

O sinal recebido começa pela passagem de um amplificador de baixo ruído (LNA). De seguida, este é convertido para uma frequência intermédia, sendo desdobrado nas suas componentes I/Q. Neste ponto, a componente analógica termina com a digitalização do sinal.

Neste sinal digital são, então, feitas algumas operações, tais como desmodulação do sinal, filtragem, correcção de erros etc. Para finalizar, a informação recebida é colocada no *buffer* de recepção, onde é acedida por protocolo SPI.

Transmissão

A nível do ramo de transmissão, o módulo começa por ir buscar informação ao *buffer* de transmissão. Seguidamente, esta irá passar por alguns blocos de tratamento os quais são responsáveis por adição de bits de sincronização, modulação etc. Depois, são criadas as componentes I/Q da informação, sendo por fim amplificadas pelo componente PA.

Configuração

Tal como foi referido, este CI possui a capacidade para funcionar em diferentes gamas de frequências. Para a realização deste trabalho foi escolhida a gama 387-464MHz, uma vez que nesta gama de frequências a ocupação do espectro é relativamente baixa quando comparada por exemplo com a gama de 2.4GHz, o que nos permite uma transmissão sem que haja grandes problemas a nivel de colisão entre dispositivos. Esta gama de frequências permite igualmente uma distância de transmissão relativamente elevada. De forma a obter o máximo rendimento do *transceiver,* é necessário fazer uma adaptação deste circuito à antena exterior para esta gama de frequências. O fabricante disponibiliza umas serie de configurações, bem como ferramentas para cálculo dos valores dos componentes electrónicos exteriores.

Neste trabalho recorreu-se a um *evaluation module,* o qual já possui o *transceiver* CC1101 colocado em circuito impresso, bem como toda a malha de adaptação. Este encontra-se ilustrado na figura seguinte. O seu esquema eléctrico encontra-se em anexo.

Figura 5.2: CC1101 *Evaluation Module [25]*

5.3 Ligação

Uma vez que estes módulos não se encontram embutidos nas placas de desenvolvimento, foi necessária a construção de placas de circuito impresso para fazer a ligação entre o módulo e as placas de desenvolvimento. Uma vez que é necessário implementar o protocolo SPI em FPGA, foi criada igualmente uma placa que serve para fazer *degub*. Esta possui num *buffer*, permitindo um isolamento em relação ao barramento SPI. Desta forma, é possível visualizar os diversos sinais sem afectar os sinais originais, uma vez que, qualquer ponta de prova possui uma capacidade parasita afectando o tempo de resposta e, deste modo, a transferência em si. Este *buffer* permite igualmente, em caso de um erro humano, como por exemplo má colocação das pontas de prova, não colocar em causa a integridade quer do módulo wireless quer das placas de desenvolvimento. Para efectuar o *debug*, foi utilizado um *logic analyzer* o qual permite a visualização simultânea de vários sinais, facilitando a construção da interface. As placas desenvolvidas encontram-se representadas na figura seguinte. O seu esquema eléctrico bem como os desenhos das placas de circuito impresso encontram-se em anexo.

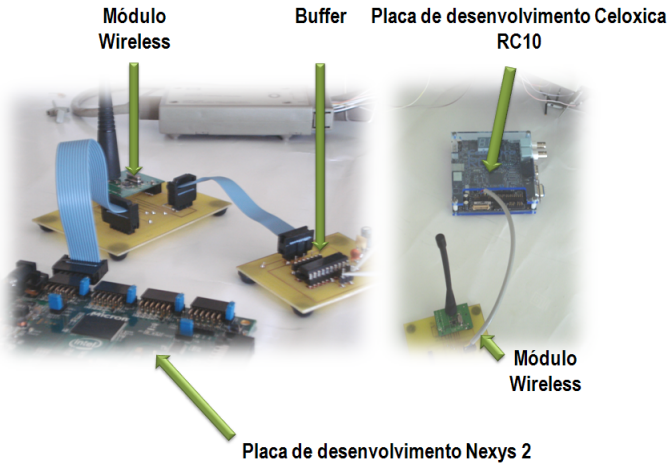

Figura 5.3: Hardware utilizado

5.4 Conceitos Fundamentais

Protocolo SPI

O protocolo SPI foi desenvolvido pela Motorola e é utilizado por uma grande quantidade de fabricantes. A sua ligação constitui uma ligação *Full-Duplex* permitindo a recepção e transmissão simultaneamente. Os dispositivos envolvidos na transferência comunicam através de uma relação *master/slave,* sendo da responsabilidade do *master* iniciar a

transferência, bem como gerar o relógio. A ligação entre dispositivos é feita a 4 fios: SCLK (Sinal de Relógio), MISO (Entrada de dados para o *master*), MOSI (Entrada de dados para o *slave*) e SS (Selecção do *slave*). Em caso de múltiplos *slaves,* deverão existir múltiplas linhas de selecção dos *slaves*.

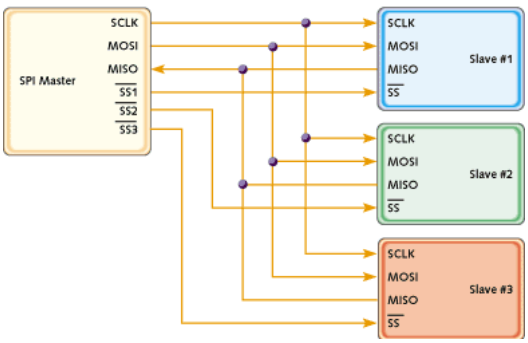

Figura 5.4: Exemplo do protocolo SPI para múltiplos *Slaves*[26]

Neste protocolo é necessário definir dois parâmetros designados por **polaridade do relógio (CPOL)** e **fase do relógio (CPHA)**. Estes definem os flancos em que a informação no barramento deve ser lida e alterada. Em caso de múltiplos *slaves* estes poderão possuir estes parâmetros diferentes pelo que é da responsabilidade do *master* reconfigurar-se para acessos diferentes.[26]

Interface SPI

Tal como já foi referido, o acesso ao CC1101 é feito através do protocolo SPI, sendo, a partir desta, feita toda a configuração do módulo wireless, assim como envio/recepção da informação a transmitir/receber.

Este possuí uma configuração a 4 fios, ou seja, utiliza 4 sinais: **MOSI**, **MISO**, **SCLK** e **CSn**. Numa transferência SPI existe sempre um *master* e um *slave*, sendo, no nosso caso, o master a FPGA e o slave o CC1101. Deste modo, os sinais de relógio bem como o *chip enable* devem ser controlados pela FPGA.

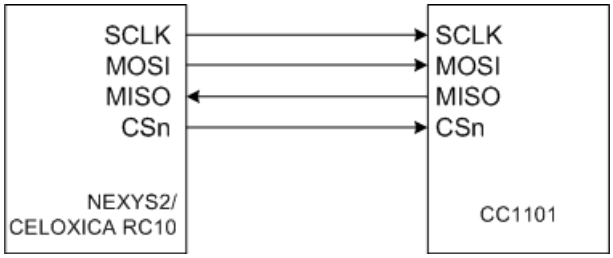

Figura 5.5: Configuração SPI a 4 fios

Configuração da interface SPI

Visto a FPGA ser o *master,* tém a responsabilidade de gerar o relógio necessário à transmissão. Para a comunicação com o CC1101, é necessário que os dados que se encontram sejam lidos na transição ascendente do relógio, enquanto a sua alteração deve ser feita na transição descendente do mesmo.

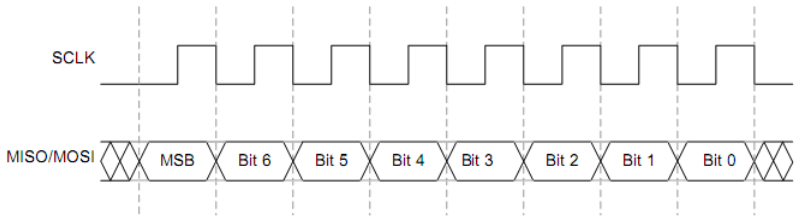

Figura 5.6: Fase e polaridade do relógio para comunicação com o CC1101[27]

Acesso por SPI

Todas as transacções através da interface SPI começam com um *byte* de cabeçalho. Este, permite definir em que registo do módulo CC1101 vamos escrever/ler e o tipo de acesso. A sua estrutura encontra-se representada na figura seguinte:

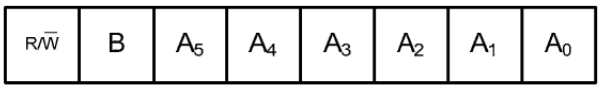

Figura 5.7: *Header Byte* [27]

- **R/W\ -** Define se a operação é de escrita ou de leitura;
- **B -** Define se o acesso é *single* ou *burst.* (Tópico abordado posteriormente);
- **A5...A0 -** Endereço do registo ao qual se pretende aceder;

O acesso ao módulo começa pela colocação do sinal CSn no nível lógico '0'. Este deverá manter-se até que a transferência termine. Quando o CSn é colocado a '0', é necessário que a FPGA espere que a linha MISO tome o valor lógico '0'. Isto indica que o cristal e as tensões de alimentação do módulo se encontram estáveis para a transferência. A partir deste momento, a transferência pode tomar lugar. A figura seguinte mostra como é feita a leitura e a escrita de um registo do módulo.

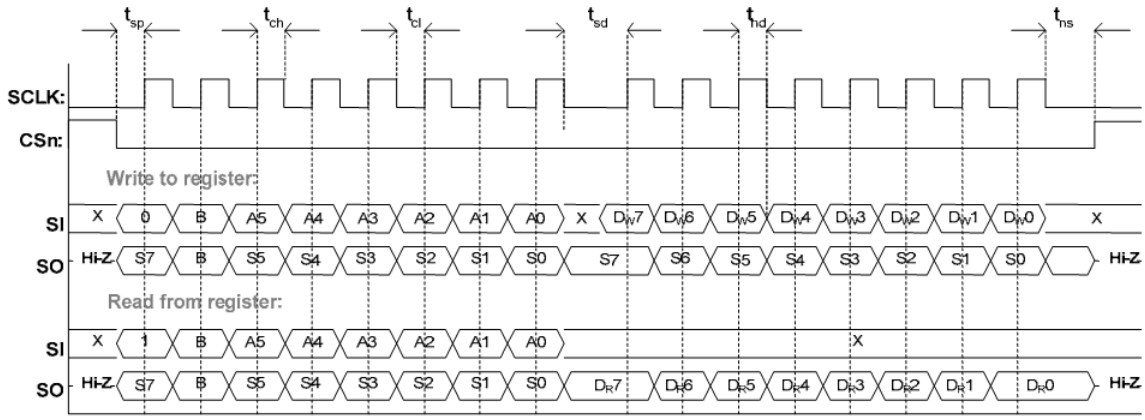

Figura 5.8: Exemplo de comunicação com o CC1101 [24]

Observando a figura anterior, é possível verificar que existem limites mínimos e máximos temporais que é necessário cumprir. Estes encontram-se em anexo diferindo para acessos *single* e *burst*.

Acesso simples (*single*)

Para fazer um acesso simples, o bit B do byte de cabeçalho deve possuir o valor lógico '0'. Após a transmissão deste, é feita a escrita ou leitura da informação dependendo do bit R/W\. Num acesso simples, após a transmissão/recepção dos dados, o *chip* fica à espera de um novo byte de cabeçalho e só depois é possível a transferência de novos dados.

Acesso *burst*

Neste caso, o bit B do *byte* de cabeçalho deverá possuir o valor '1'. Neste cabeçalho é enviado o endereço inicial que se pretende escrever/ler, sendo este incrementado automaticamente pelo CC1101 para o endereço seguinte. Deste modo, não existe a necessidade de um novo *byte* de cabeçalho. Este acesso só deve ser utilizado quando existe a necessidade de aceder a endereços consecutivos do CC1101. Para terminar um acesso *burst* apenas é necessário colocar a linha CSn com o valor lógico '1'.

Command Strobes

Command Strobes são um tipo especial de comandos que desencadeiam uma série de operações no CC1101. Estes, são acedidos através dos endereços 0x30 até 0x3D. Para despoletar uma das operações, apenas é necessário fazer uma leitura/escrita para estes registos. Esta leitura/escrita deverá ser feita, obrigatoriamente, com o bit B do byte de cabeçalho a '0'. Estes comandos permitem colocar o módulo, por exemplo, em estado de recepção, transmissão etc. Em anexo, encontra-se uma tabela com todos os comandos disponíveis. Caso o bit B seja colocado a '1' em qualquer um destes endereços, serão acedidos não os *Command Strobes* mas registos de *Status*. Estes registos contêm informações importantes, tais como o número de bytes na FIFO de transmissão e recepção, estado actual do módulo etc.

Status Byte

Quando um *byte* de cabeçalho, de dados ou um *command strobe,* é enviado pela interface SPI, um *byte* designado por *status byte* é enviado pelo *chip* que possui informação útil para a FPGA/Microcontrolador. A sua estrutura encontra-se representada na figura seguinte.

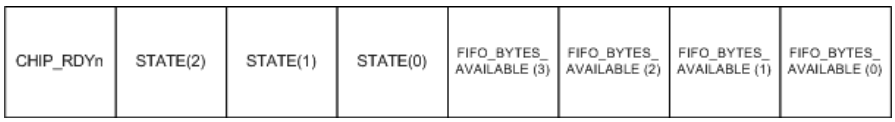

Figura 5.9: *Status Byte*

- **CHIP_RDYn -** Este toma o valor lógico '0' enquanto as tensões de alimentação e o cristal estabilizarem, caso contrário '1'.
- **STATE[2:0] -** Estes indicam o estado actual da máquina de estados que controla o CC1101. Ex: Quando 001 o módulo wireless encontra-se no estado de recepção.
- **FIFO_BYTES_AVAILABLE[3:0] -** Caso seja feita uma leitura, este campo indica o número de bytes na FIFO de recepção. Caso seja feita uma escrita, este indica o número de bytes livres na FIFO de transmissão;

Mapeamento do módulo CC1101

O módulo possui 47 registos de configuração. Estes registos permitem definir parâmetros de grande relevância, tais como: modulação, taxa de transmissão, largura de filtros etc. Estes encontram-se situados nos endereços 0x00 até 0x2E. Para além destes, existem mais dois registos que se encontram situados nos endereços 0x3E e 0x3F.

No primeiro, situa-se o registo de controlo de potência do PA. O segundo, tem como objectivo aceder às FIFOs de transmissão e recepção. Este acesso vai ser discutido no tópico seguinte.

Entre os endereços 0x30 e 0x3D, situam-se os *commands strobes* e os registos de *status* (tópico já abordado anteriormente).

Em anexo, encontra-se uma ilustração com todos estes registos e como devem ser acedidos.

Acesso às FIFOs de transmissão e recepção

O acesso a estes elementos é feito através do endereço 0x3D, tanto para escrita da FIFO de transmissão como para leitura da FIFO de recepção. Para efectuar uma leitura da FIFO de recepção, apenas é necessário indicar no byte de cabeçalho que se pretende fazer uma leitura e indicar o endereço 0x3D. Esta leitura pode ser feita através de um acesso simples ou *burst*. É de notar que esta leitura só deve ser feita quando esta possuir informação. O valor do número de bytes que se encontram na FIFO de recepção pode ser lido através do *status byte* ou lendo o registo *status* que se encontra no endereço 0x3B.

Caso seja indicado no *byte* de cabeçalho uma operação de escrita, vamos estar a aceder, não à FIFO de recepção mas de transmissão. Os dados podem ser igualmente escritos das duas formas de acesso: *single* e *burst*. É preciso ter, igualmente, em conta se a FIFO não se encontra cheia, de forma a não perder informação. Esta informação pode retirada a partir do valor de bytes na FIFO indicado pelo *status byte* ou, então, acedendo ao registo *status* que se encontra no endereço 0x3A.

Formato do pacote de informação

Em cada transmissão é enviado um pacote, possuindo este campos extra à informação útil que queremos transmitir.

São possíveis no módulo CC1101 configurar os seguintes campos:

- *Preamble*;
- Tamanho da palavra de sincronização;
- Tamanho em bytes dos dados úteis (Opcional);
- Endereço de destino (Opcional);
- *Payload*;
- *Cyclic Redundancy Check* (Opcional);

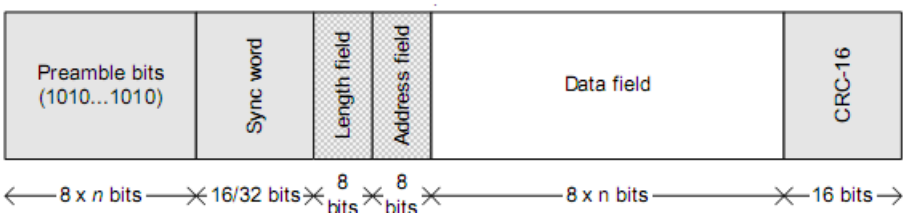

Figura 5.10: Formato de pacote[24]

O campo *preamble* consiste numa sequência de uns e zeros, sendo este campo colocado automaticamente pelo CC1101 na transmissão e removido automaticamente na recepção. É possível configurar o número mínimo de bits que são enviados. Este campo, quando o módulo se encontra no estado de transmissão, emite continuamente até que seja detectada informação na FIFO de transmissão. Quando detectada informação nesta, o CC1101 começa a enviar a palavra de sincronização. Este possuí uma tamanho configurável pelo programador (16/32 bits), sendo colocado automaticamente na transmissão e removido na recepção.

O módulo CC1101 permite tamanhos de pacote fixos ou variáveis. Caso o tamanho seja variável, é enviado o campo *length field*. Este campo tem de ser escrito pelo programador e deve conter o número de bytes de informação útil que são enviados no pacote. Esta informação deve ser colocada na transmissão e retirada na recepção pelo programador.

Tal como foi dito, é possível criar uma rede a partir destes módulos. Caso essa funcionalidade seja activada, é igualmente necessário que o programador envie o endereço destino, sendo este enviado no campo *Address Field*.

De seguida, é enviada a informação "útil". Esta pode ter tamanho fixo ou variável conforme tenha sido configurado o módulo wireless.

Para finalizar, são enviados dois *bytes* que permitem testar se houve erros na transmissão, permitindo sinalizar ao programador se a informação recebida é ou não válida. Esta informação é colocada automaticamente na transmissão, mas necessita de ser retirada pelo programador na recepção.

5.5 Implementação da interface para o módulo CC1101

Neste trabalho foi implementada toda a interface para o módulo CC1101. Serão agora apresentados todos os blocos construídos, que constituem a *stack* protocolar desenvolvida.

5.5.1 Gerador do SCLK

Uma vez que nesta ligação SPI a FPGA se encontra definida como *Master,* é necessário que esta gere o relógio necessário à transferência. Segundo as informações recolhidas do fabricante, a frequência máximo que o relógio pode ter é de 10MHz. No entanto, este indica, igualmente, que se tal frequência for utilizada, é necessário um tempo de espera entre cada *byte* transferido, no mínimo de 100ns.

Figura 5.11: SCLK a 10MHz [27]

No entanto, segundo o fabricante, pode usar-se uma frequência de relógio até 6.5MHz não sendo, neste caso, necessário um tempo de espera.

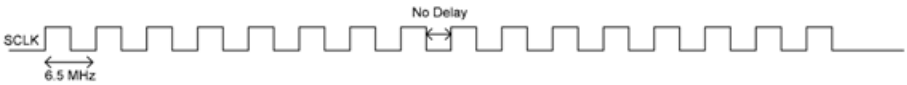

Figura 5.12: SCLK a 6.5MHz [27]

Uma vez que a taxa de transferência utilizada será na ordem dos Kbits/s, não existe a necessidade da utilização da frequência de relógio de 10MHz, uma vez que o acesso ao módulo irá ser feito, mesmo utilizando 6.5MHz muito mais rápido que a taxa de transferência entre módulos, facilitando, igualmente, o bloco na FPGA que gera o relógio (SCLK).

O bloco construído possui 3 entradas: **Clock** (proveniente do cristal da placa de desenvolvimento), ***Reset*** e ***Enable***. Como saídas, este possuí apenas o relógio para a ligação SPI: **SCLK**. Este bloco possuí igualmente um parâmetro genérico *(Divider)*, que pode ser alterado permitindo alterar a frequência do SCLK.

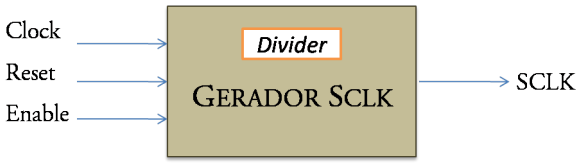

Figura 5.13: Gerador SCLK

Para a construção deste bloco, foi utilizada uma máquina de estados finitos e segue o diagrama de fluxo de sinal apresentado na figura seguinte.

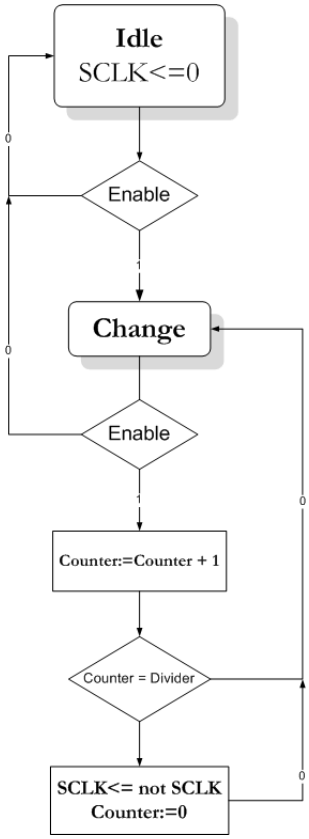

Figura 5.14: Diagrama defluxo de sinal do bloco Gerador do SCLK

Observando a figura anterior, é possível verificar que a iniciação do relógio é despoletada por camadas superiores. Quando o sinal *enable* se encontra activo, o SCLK é gerado sendo a sua frequência alterada pelo valor genérico *divider*. A forma como é o relógio dividido, consiste num contador que, quando atinge o valor *divider,* o relógio SCLK troca de sinal lógico. Para as placas de desenvolvimento utilizadas, a frequência máxima do relógio do cristal destas é de 50MHz, pelo que foi utilizado um valor de 9 para divisão do relógio de entrada conseguindo-se, desta forma, uma frequência para o SCLK menor a 6.5MHz (valor já justificado anteriormente).

5.5.2 Controlador de transferência de um *byte*

Esta camada encontra-se imediatamente acima do bloco Gerador de SCLK. Esta, tem como objectivo receber/transmitir um byte de informação gerindo o sinal de relógio SCLK e as linhas MISO, MOSI e CSn.

A figura seguinte, mostra um esquema deste componente:

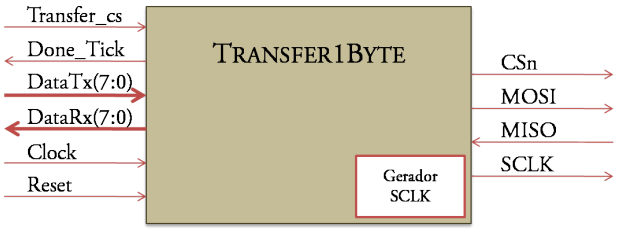

Figura 5.15: Controlador de transferência de um byte

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte, sendo este executado por uma máquina de estados finitos simples:

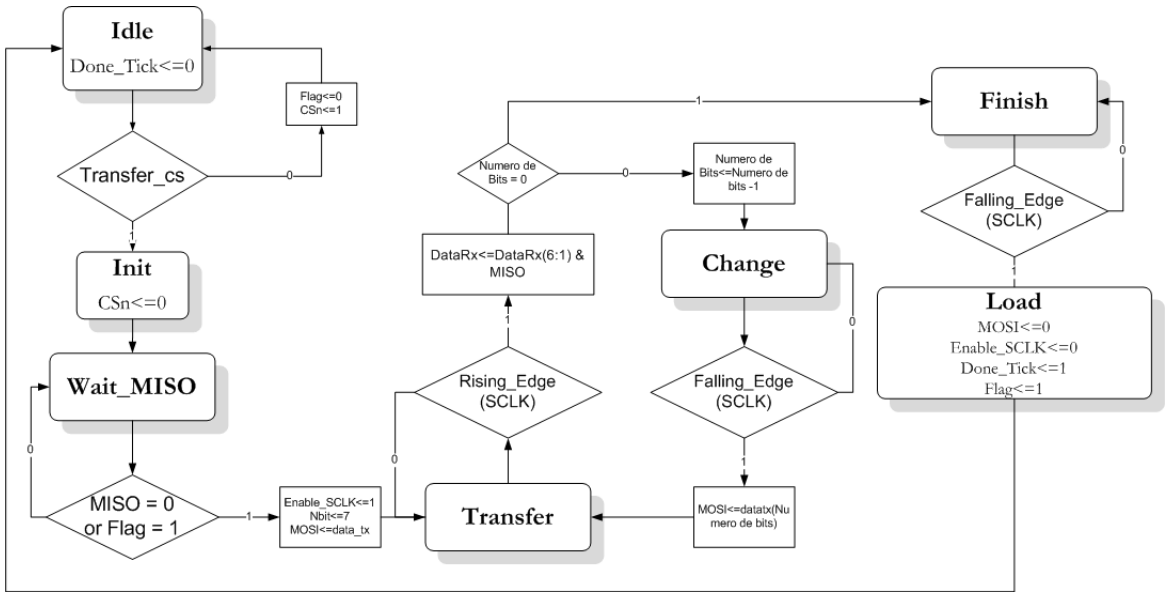

Figura 5.16: Diagrama de fluxo de sinal do bloco Transfer1Byte

Esta, encontra-se, por defeito, no estado *Idle,* até que seja feito um pedido por um elemento superior, sendo este pedido feito através do sinal de entrada *transfer_ cs*. Quando activo, a máquina de estados transita para o estado *Init* onde coloca o CSn a zero esperando que o CC1101 coloque a zero a linha MISO (*CSn_ Chip*) indicando, desta forma, que a transferência pode tomar lugar. No entanto, esta não é a única condição, isto é, caso a linha CSn já se encontre a zero e seja necessário a transferência de um novo byte, não existe a necessidade de esperar pela linha MISO com o valor zero. Para indicar que o byte a transferir não é o primeiro (Byte de cabeçalho) existe uma flag, a qual toma o valor '1' quando a transferência do primeiro byte é concluída. Então, caso um destes casos aconteça, o SCLK é activo e a linha MOSI fica com o valor do bit mais significativo. Transita, deste modo, para o estado *Transfer,* o qual fica à espera da próxima transição ascendente do relógio (SCLK). Neste ponto, é lida a informação

da linha MISO, sendo guardada para o vector DataRx. De seguida, é testada a variável Numero de bits. Se esta contiver o valor 0, é porque o *byte* já foi enviado, transitando para o estado *Finish*. Neste, a flag irá possuir o valor lógico um, sendo igualmente accionado um sinal de *done,* indicando à camada superior que a transferência de um byte se encontra concluída. Caso contrário, se ainda não foi transmitido o byte (Numero de Bits diferente de 0), a máquina transita para o estado *Change*. Esta espera pela próxima transição descendente do SCLK, alterando nesta a informação que se encontra na linha MOSI, regressando de seguida para o estado *Transfer,* repetindo-se o ciclo até que a transferência do *byte* se encontre concluído. Quando a FSM regressa ao estado *Idle*esta testa se o sinal *transfer_ cs* ainda se encontra activo. Se tal acontecer, o ciclo repete-se sem a espera pela linha MISO tomar o valor 0. Caso contrário, a flag é colocada a zero indicando que o próximo byte será um byte de cabeçalho e colocando a linha CSn com o valor lógico '1'.

5.5.3 Controlador da transferência SPI

Tal como foi dito na secção "conceito introdutórios", a transferência SPI para o módulo CC1101 é inicializada por um byte de cabeçalho seguida de um byte de dados em caso de escrita. Ao mesmo tempo são recebidos *status bytes* ou dados no caso de uma leitura. Este bloco utiliza o anterior para a gestão da transferência de dados fazendo-o apenas por acesso simples. Este recebe o cabeçalho e os dados a transmitir, (caso seja leitura o segundo byte possui um valor genérico) chama as camadas anteriores e separa automaticamente a informação que recebe da interface SPI para as respectivas saídas: *status byte* e informação de registos/Dados. O seu bloco encontra-se representado na figura seguinte:

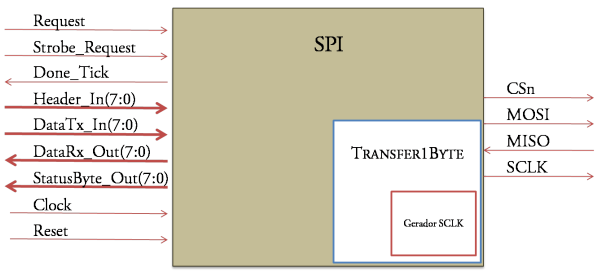

Figura 5.17: Bloco SPI

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte:

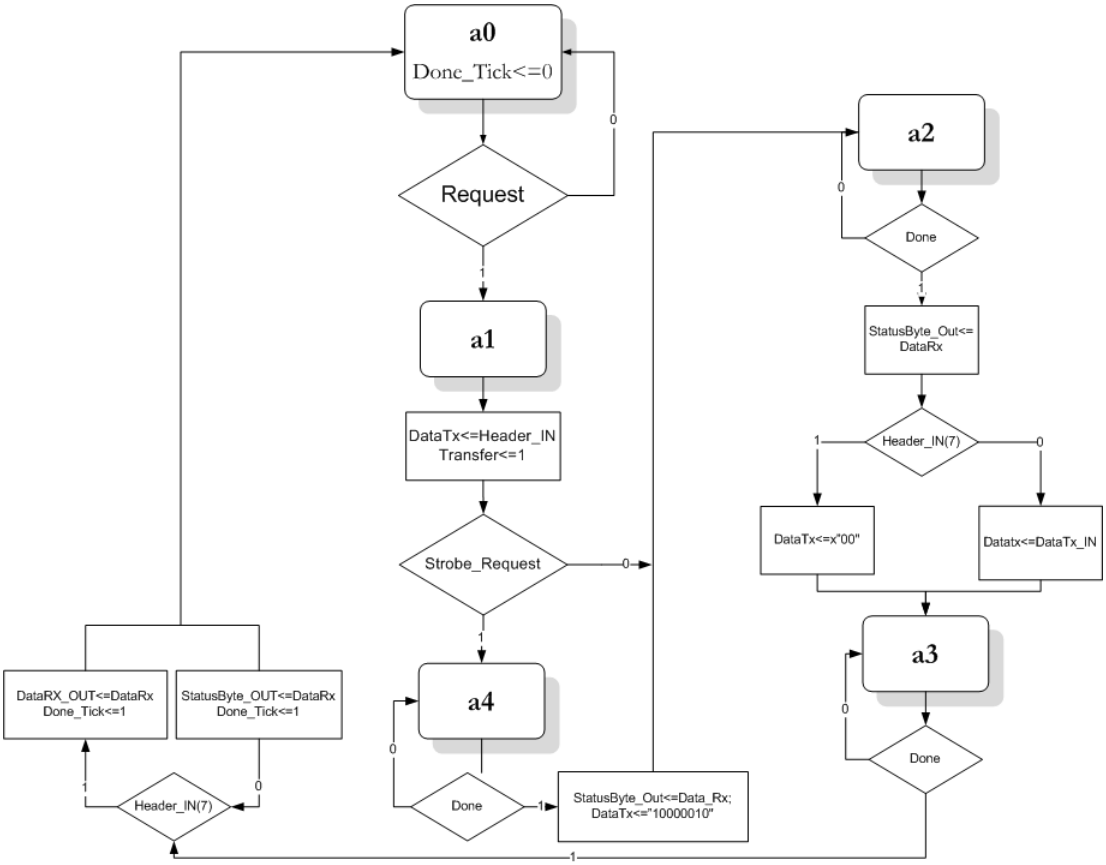

Figura 5.18: Diagrama de fluxo de Sinal do bloco SPI

Este bloco é controlado por uma FSM, começando no estado a0. Esta mantém-se neste, até que seja feito um pedido da camada superior a partir do sinal *request*. Quando tal acontece, a FSM transita para o estado a1 onde é feito um pedido de transmissão do *header byte*.

Caso o sinal *strobe_ request* se encontre com o valor lógico ", a máquina passa directamente para o estado a2, ficando à espera que o *header byte* seja transmitido. Quando é recebido o sinal de *done,* o valor do sinal de saída *StatusByte_ Out* é actualizado, uma vez que na transmissão do *header byte*é recebido na linha MISO o *StatusByte*. De seguida, é enviado o segundo byte. Caso seja feita uma leitura, o segundo byte possui o valor genérico 0x00. Caso contrário, ou seja, numa operação de escrita, é transmitido o valor proveniente da camada superior (*DataTx_ In*).

Após o pedido de transmissão a FSM fica à espera do sinal de *done*. Quando este toma o valor lógico '1', em caso de uma operação de leitura o sinal de saída *DataRX_ OUT* é actualizado com a informação proveniente da linha MISO. Caso seja feita uma operação de escrita, o sinal proveniente na linha MISO é novamente o *Status Byte,* pelo que o sinal de saída correspondente será novamente actualizado. Depois das

respectivas atribuições, ao sinal *Done_ Tick* é atribuído o valor lógico '1', indicando à camada superior que a transmissão se encontra concluída. Para finalizar, a FSM regressa ao estado a0.

Caso seja efectuado um pedido para um *command_ strobe,* a FSM irá passar não do estado a1 para o a2, mas para o estado a4. Neste, é feita uma espera que a transmissão do *header byte* se encontre concluída. Segundo o fabricante, quando se pretende enviar um *command_ strobe,* deve ser feita uma leitura arbitrária após o envio do primeiro *header byte.* Para tal, é enviado, de seguida. um segundo *header byte* para um endereço genérico, transitando, depois, a máquina para o estado a2 e sendo feita uma leitura normal.

5.5.4 Controlador da transferência de dados

A construção deste módulo teve como objectivo receber a informação de um pedido vindo de uma camada superior para uma transferência reunindo-a e distinguindo automaticamente em conformidade com os parâmetros recebidos se o pedido se trata de um *commnand Strobe* ou uma simples leitura/escrita para um registo Esta informação é de seguida enviada para o bloco SPI. A figura seguinte, mostra um diagrama de blocos deste componente.

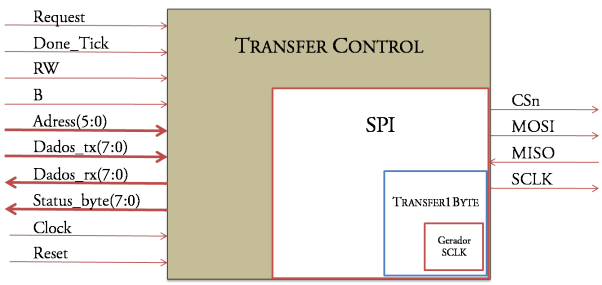

Figura 5.19: Bloco Transfer Control

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte.

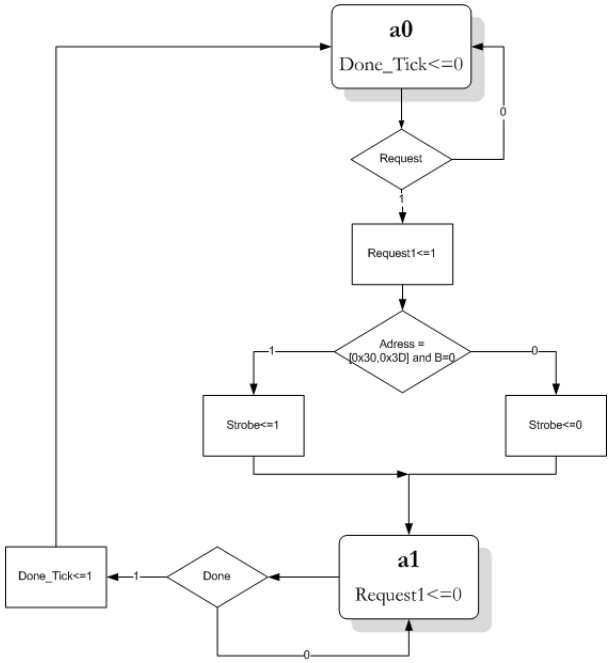

Figura 5.20: Diagrama de fluxo de sinal do bloco TransferControl

Este bloco foi construído através de uma máquina de estados finitos, encontrando-se esta por defeito no estado a0. Quando é feito um pedido, é verificado o endereço de entrada (*Adress(5:0)*) e o bit *B*. Em conformidade com estes valores, este bloco detecta automaticamente se se trata de um *command_strobe* ou de um acesso a um registo, activando a linha *Request Strobe* do bloco SPI em caso afirmativo. É então, feito um pedido à camada SPI para a transferência da informação, passando de seguida para o estado a1. Neste, a máquina de estados finitos espera de uma confirmação de envio a qual é feita através do sinal *Done*. Quando este valor possui o valor lógico '1', a transferência encontra-se concluída fazendo regressar a FSM ao estado a0, podendo, neste momento, receber outro pedido.

5.5.5 Configurador do módulo CC1101

Estando o protocolo SPI implementado, é necessário agora configurar o módulo wireless, ou seja, configurar todos os registos para que este se adapte em conformidade com as características escolhidas. Ao bloco responsável por esta função designou-se de Configurador. Os valores dos registos que devem ser configurados encontram-se, por sua vez, numa memória ao qual este componente acede de forma sucessiva, enviando esta informação para as camadas inferiores. A figura seguinte, mostra um diagrama de blocos deste componente.

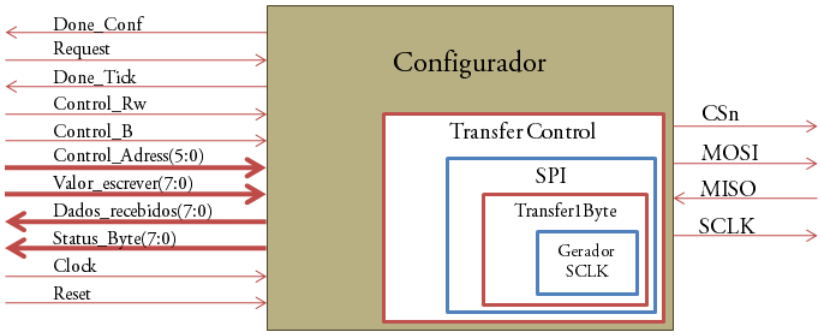

Figura 5.21: Componente Configurador

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte:

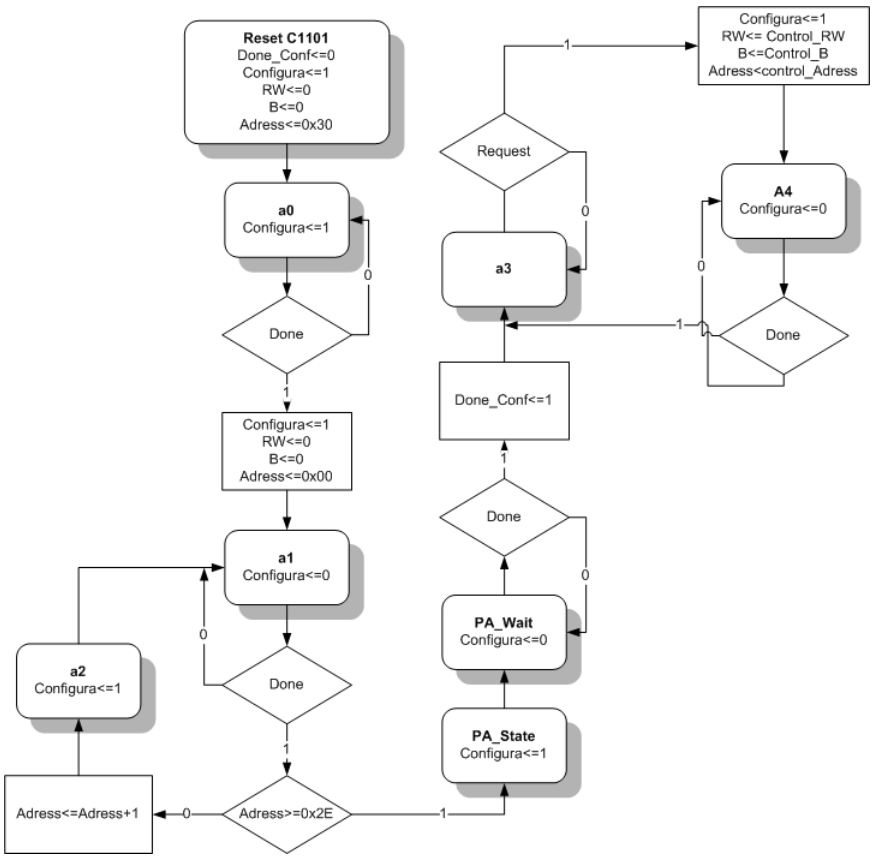

Figura 5.22: Diagrama de fluxo de sinal do bloco configurador

Observando a figura anterior, pode-se observar que a máquina de estados finitos que controla este componente começa por efectuar um reset ao módulo CC1101, recorrendo, para isso, a um *command_strobe* que se encontra no endereço 0x30. De seguida, este transita para o estado a0 o qual fica à espera da confirmação da conclusão do comando de reset. Tendo a certeza, neste momento, que o chip se encontra no seu estado inicial, começa a configuração de todos os registos, por ordem crescente de endereços. Este

ciclo repete-se pelos estados a1 e a2.Quando é atingido o endereço 0x2E, a configuração crescente de endereços pára, uma vez que a partir deste endereço, encontram-se os registos de *statuse* os *command strobes*. É feito, então, um salto para o endereço 0x3E, onde se encontra o registo que programa a potência do sinal transmitido. A FSM passa, deste modo, para o estado PA_State, onde é feito o pedido à camada inferior para a configuração deste registo, transitando de seguida para o estado PA_wait. Neste, existe uma espera pelo sinal de *done* da camada inferior, indicando que o registo já se encontra configurado.

Conclui-se, desta forma, a configuração dos registos do CC1101, sendo esta conclusão indicada à camada superior pela activação do sinal *Done_ conf.*

Após a fase de configuração, a máquina de estados mantêm-se no estado a3, o qual recebe pedidos e os reencaminha para as camadas inferiores, transitando para o estado a4, onde fica à espera da indicação da conclusão da transferência, regressando, novamente, para o estado a3 após a sua conclusão e esperando pelo próximo pedido.

5.5.6 Controlador do módulo CC1101

Após a configuração do módulo wireless, é necessário efectuar o seu controlo, isto é, colocá-lo em modo de transmissão e recepção, bem como gerir a informação nas FIFOs dividindo a informação em campos para utilização nas camadas superiores. Este bloco permite, a camadas superiores, obter a completa abstracção com as características do módulo, isto é, as camadas superiores apenas se preocupam em enviar/receber informação, não necessitando de saber pormenores, tais como o estado do módulo, uma vez que, esta gestão é feita automaticamente pelo bloco em causa.

Trama construída pelo componente controlador

Tal como foi referido anteriormente, o módulo CC1101 possuí uma trama com uma determinada estrutura. No entanto, um programador pode definir uma sub-trama, bastando para isso dividir o espaço reservado para dados, criando, desta forma, uma trama que só o programador conhece e só este consegue separar a informação correctamente. Visto a aplicação em causa ter como objectivo enviar informação para diversos componentes, existe a necessidade de criar um campo, o qual vamos designar por ***identifier***. Este, terá o tamanho de $1byte$ e, conforme o valor indicado neste campo, é possível distinguir para que componente a informação transferida remotamente se destina. Definiu-se, igualmente, que o tamanho dos dados úteis seria de $4bytes$. A figura seguinte, mostra a trama que é enviada/recebida por cada módulo wireless.

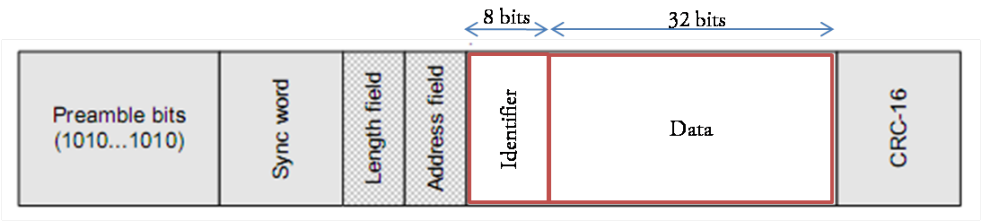

Figura 5.23: Sub-trama criada pelo componente controlador

Implementação de uma ligação Bidireccional

Um dos objectivos para este trabalho é permitir uma ligação bidireccional, isto é, os módulos wireless devem conseguir transmitir e receber. Para que isto seja possível, é necessário que o módulo se encontre sempre no modo de recepção, excepto quando pretende transmitir. Esta funcionalidade é configurável no módulo CC1101, ou seja, este permite definir que, após a transmissão, o módulo passa automaticamente para o modo de recepção. Este método, no entanto, necessita de alguma robustez, uma vez que, existe sempre a probabilidade de colisão de informação. Esta pode ser reduzida, em parte, activando outra funcionalidade deste módulo: CCA (*Clear Channel Assessment*). Esta funcionalidade permite saber se existe alguma informação a ser recebida. Assim, caso seja feito um pedido de transmissão e a flag CCA se encontre activa este não irá ser processado, sendo por isso necessário tentar voltar novamente mais tarde. Este método é bom e diminui a probabilidade de colisão, mas não o combate a 100%, pelo que foi construído um componente que implementa um protocolo, diminuindo a probabilidade de perda de informação. Este será apresentado em tópico posterior.

Diagrama de blocos e diagrama de fluxo de sinal

O diagrama de blocos deste componente encontra-se representado na figura seguinte.

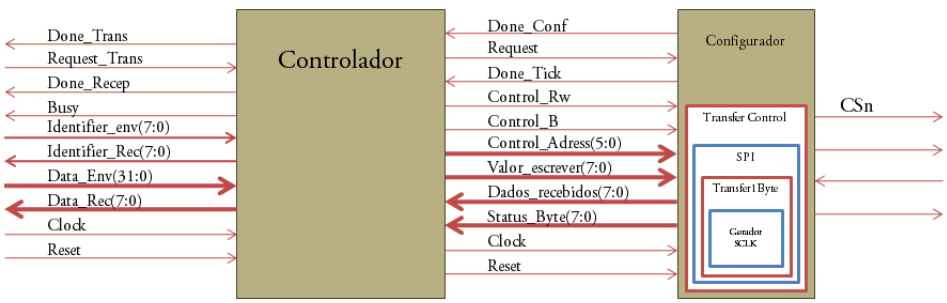

Figura 5.24: Componente Controlador

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte.

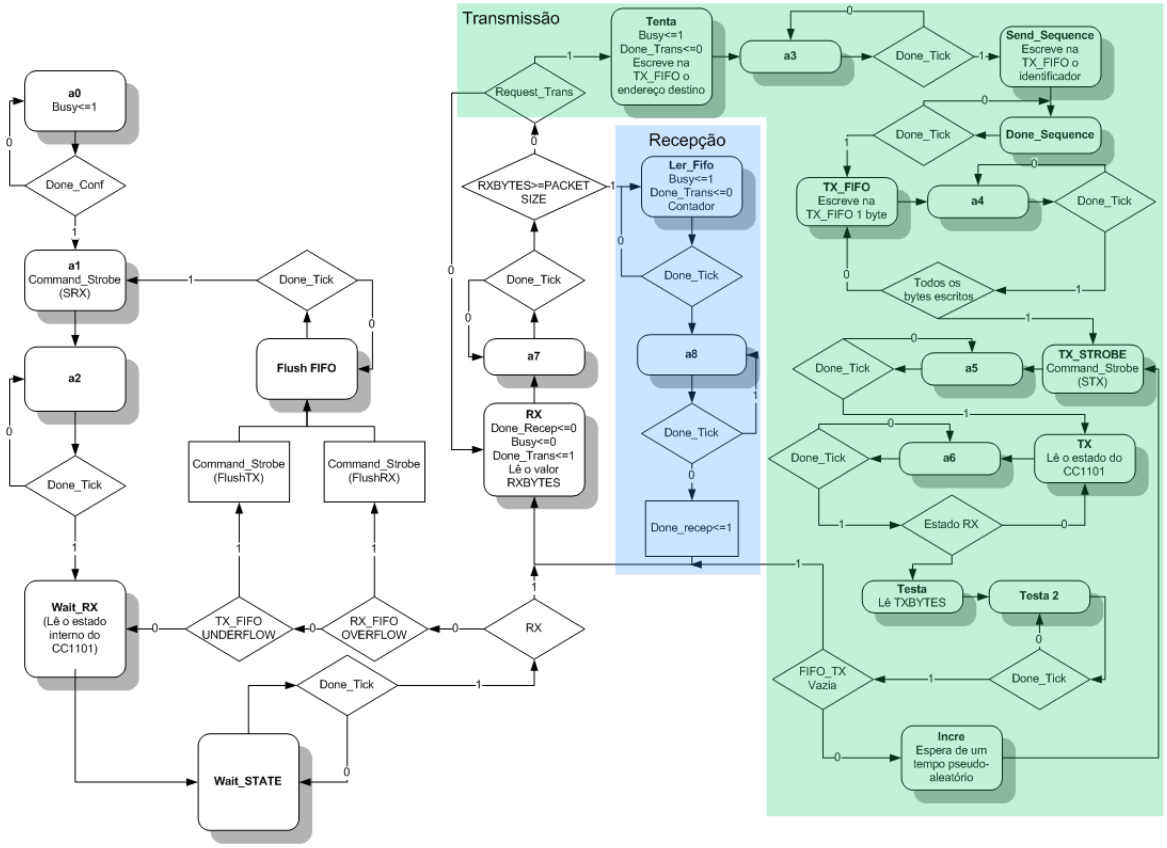

Figura 5.25: Diagrama de fluxo de sinal do bloco controlador

Este bloco encontra-se implementado, mais uma vez, a partir de uma FSM simples. Esta começa no estado a0, no qual espera que a configuração termine. Quando recebida a confirmação, este tenta colocar o módulo em modo de recepção através de um *command_strobe* (SRX), passando de seguida para o estado a2, esperando que o comando esteja concluído. De seguida, é lido o estado do módulo. Uma vez que existem tempos que não podem ser controlados pelo programador, por vezes o módulo pode atingir os estados de *underflow* da FIFO de transmissão ou *overflow* da FIFO de recepção. Neste caso, estes dados são limpos e é pedido, novamente, que o módulo passe para o estado de recepção. Quando tal acontece, a FSM transita para o estado RX. Neste, é feito, constantemente, um pedido para saber o número de bytes que se encontram na FIFO de recepção. Caso este número seja maior ou igual ao valor do tamanho do pacote, é feita a leitura da FIFO, retirando a informação e enviando-a para a camada superior, indicando a esta que existe nova informação, recebida através do sinal *Done_Recep*. Caso o número de bytes na FIFO de recepção seja inferior ao tamanho do pacote, é testado se existe algum pedido para transmissão. Em caso afirmativo, começa o processo de transmissão e segue os seguintes passos:

1. Enviado para a FIFO de transmissão o endereço destino pretendido;

2. Enviado para a FIFO o campo *identifier* (identificador);

3. Enviado para a FIFO os dados;

4. Enviado um *command_strobe* para o módulo transmitir;

5. É lido o estado em que o módulo ficou até que volte ao modo de recepção.

6. É lido o número de bytes que se encontram na FIFO de transmissão. Caso este valor seja não nulo, quer dizer que o canal se encontrava ocupado quando na tentativa de transmissão. Neste caso, é feito um tempo de espera pseudo-aleatório (o que diminui as probabilidades de uma nova ocupação do canal) repetindo-se, ao fim deste, os passos 4,5 e 6 até que seja enviada a informação. Caso o número de bytes seja igual a zero, a informação foi transmitida retornado a máquina para o estado RX e repetindo-se todo o ciclo de operações.

5.5.7 Controlador de transmissão

De forma a diminuir a complexidade da interface para a transmissão de dados, foi o construído o módulo ao qual designamos por Controlador de transmissão. Este irá permitir criar alguma abstracção da interface do módulo anterior. O diagrama de blocos deste componente encontra-se representado na figura seguinte.

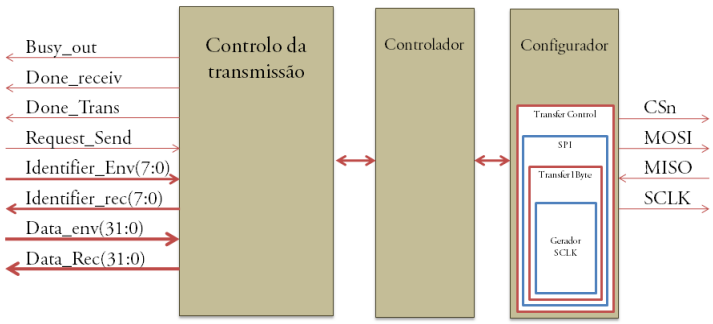

Figura 5.26: Diagrama de blocos do sistema incorporando o controlo de tranmissão

O respectivo diagrama de fluxo de sinal encontra-se representado na figura seguinte.

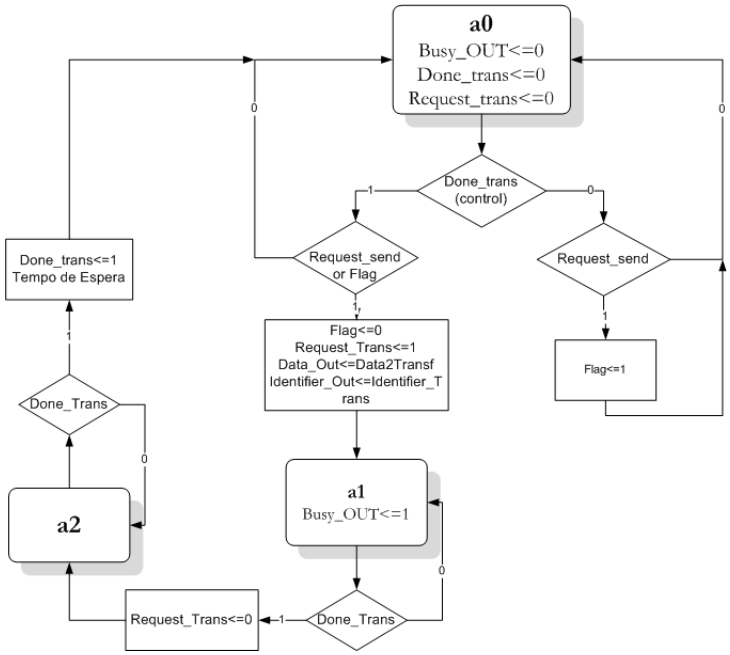

Figura 5.27: Diagrama de fluxo de sinal do controlador da transmissão

A máquina de estados finitos que controla este bloco inicia no estado a0. Neste, são colocados, por defeito, a zero todos os sinais que invocam uma transferência à camada inferior. Neste estado, é verificado um sinal do módulo controlador indicando se o módulo está pronto para receber informação para transmitir (sinal *Done_ Trans*). Se tal acontecer e for feito um pedido, este pedido é executado imediatamente, caso contrário, se o módulo não estiver pronto mas for feito um pedido, este é guardado sendo activada uma flag indicando que mal o módulo esteja pronto este pedido deve ser atendido.

Quando o módulo atende o pedido, este activa o sinal *Request_ Trans,* desencadeando todo o fluxo de transmissão no módulo controlador e fazendo transitar a máquina de estados para o estado a1. Neste, é activado um sinal de *Busy* à camada superior, indicando que o módulo se encontra ocupado. Neste estado, é igualmente testado o sinal *Done_ Trans* que, quando tomar o valor zero, indica que o módulo anterior já se encontra a processar o pedido, desactivando, assim, o sinal *Request_ Trans,* ou seja, a indicação para atender um pedido. A máquina transita, de seguida, para o estado a2 onde espera a conclusão do pedido. Visto um dos objectivos ser uma ligação bidireccional, quando o sinal de conclusão do pedido é activo foi introduzido um tempo de espera, de forma a libertar o canal RF permitindo, deste modo, que o segundo módulo wireless tenha a possibilidade de enviar informação. No final deste tempo, todo o ciclo se repete.

5.5.8 Controlador do protocolo de comunicação

Tal como foi dito anteriormente, existe sempre a possibilidade de perda de pacotes. Esta pode ser reduzida graças à utilização do indicador CCA. No entanto, isto não garante que a informação chega ao seu destino, uma vez que, pode haver uma colisão quando os dois tentam enviar ao mesmo tempo. Para reduzir a probabilidade de perca de pacotes, foi implementado um protocolo simples baseando-se em pacotes de *acknowledge*. A figura seguinte, mostra um exemplo do protocolo implementado.

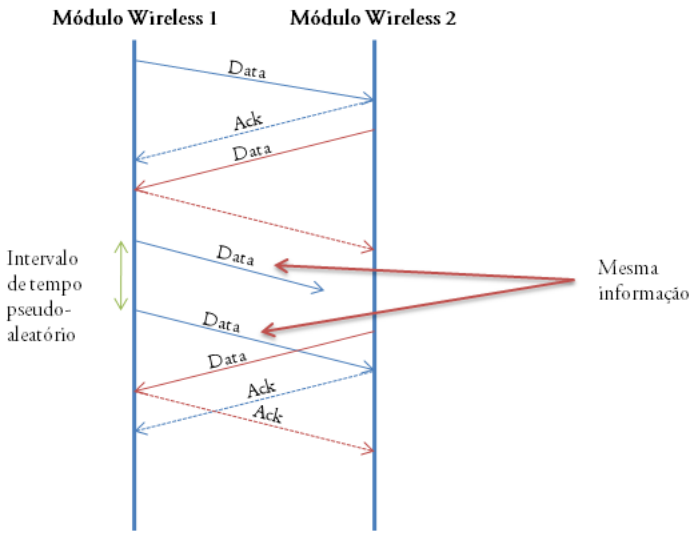

Figura 5.28: Diagrama ilustrativo do protocolo desenvolvido

Este protocolo funciona da seguinte maneira:

1. O pacote é enviado. Neste momento, um *timer* com tempo pseudo-aleatório começa a incrementar. Este serve para que, em caso da não recepção de um pacote *acknowledge*, o pacote seja retransmitido passado um determinado tempo, tempo esse que deverá possuir um valor mínimo, uma vez que é necessário que a informação seja escrita no módulo que transmite, seja propagada até ao seu destino, e seja enviado um pacote de *acknowledge* possuindo este, igualmente, um tempo de escrita no módulo e um tempo de propagação no canal.

2. O módulo que envia o pacote fica à espera da recepção de um pacote *acknowledge* identificado através do campo *identifier*. Este, no caso de um *acknowledge*, possui o valor 0x00,pelo que não deverá ser utilizado para o envio de dados "úteis".

3. Caso o pacote *acknowledge* seja recebido, o módulo pode enviar o próximo pacote. Caso contrário, quando o *timer* atingir o valor pseudo-aleatório amostrado

quando no envio, o pacote é retransmitido repetindo-se este ciclo. É de notar que, quando um dos módulos fica à espera do pacote *acknowledge*, nada o impede de não enviar pacotes *acknowledge* para o outro módulo.

O diagrama de blocos deste componente encontra-se representado na figura seguinte:

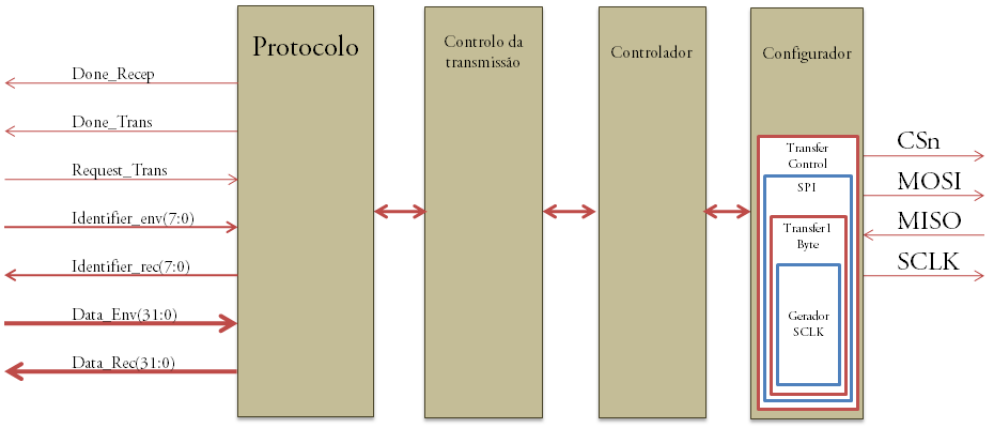

Figura 5.29: Componente Protocolo

Podemos dividir este componente em dois diagramas de fluxo de sinal: de recepção e transmissão.

Recepção

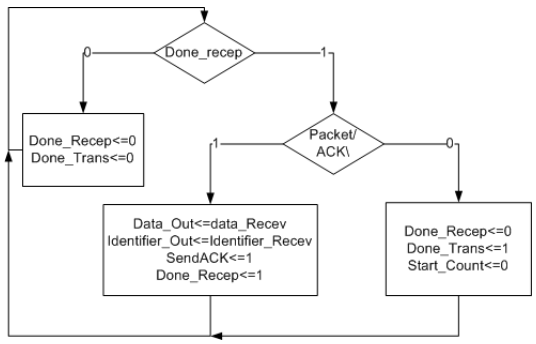

Figura 5.30: Diagrama de fluxo de sinal da parte da recepção

O controlo da recepção é implementado através de um simples processo, ao contrário dos restantes componentes implementados com recurso a máquina de estados finitos.

Neste processo, começa-se pela análise do sinal *done_recep* proveniente do módulo anterior. Caso este se encontre com o valor lógico '0', não é feita nenhuma operação, mantendo-se os sinais *Done_recep* e *Done_trans* iguais a zero, ou seja, indicando à camada superior que nada foi recebido nem transferido. Caso seja recebido algum

pacote, é avaliado o campo *identifier*. Caso este possua o valor 0x00, indica que a informação recebida se trata de um sinal *acknowledge*, pelo que a informação transmitida foi recebida pelo destino, sendo activado o sinal *Done_ trans* indicando que o pacote foi transmitido. Neste momento, o *timer* pára de contar e é reiniciado. Caso o pacote recebido seja um pacote de dados, é indicada à camada superior que recebeu um pacote activando o sinal *Done_ Recep*. É, igualmente, activado o sinal *SendACK* indicando à parte responsável pela transmissão de pacotes que deve enviar um pacote *acknowledge*.

Transmissão

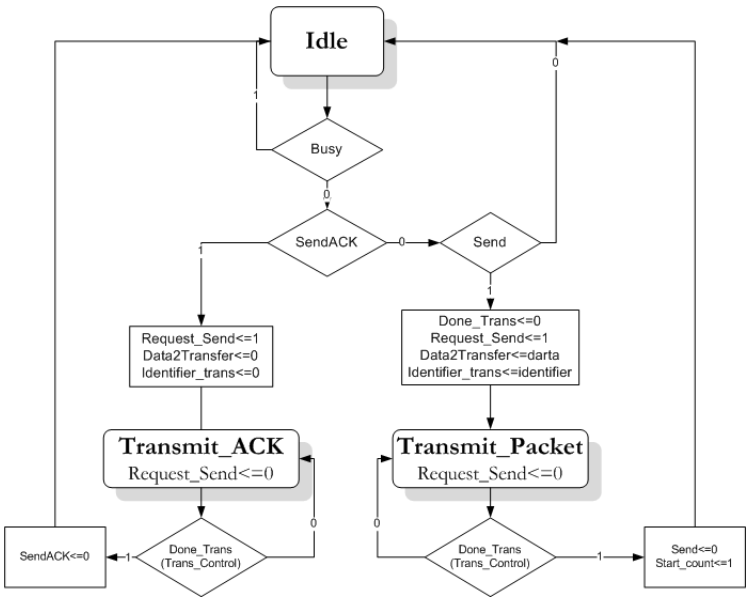

Figura 5.31: Diagrama de fluxo de sinal da parte da transmissão

O controlo da transmissão, ao contrário da recepção, é efectuado por uma máquina de estados finitos simples.

Esta encontra-se por defeito no estado *Idle*. Neste estado, é testado se o módulo se encontra ocupado, sinal este proveniente da camada inferior. Quando este se encontra livre, é testado se existe algum pacote *acknowledge* que deve ser enviado. Se tal acontecer, é feito um pedido à camada inferior para transferência de um pacote com o campo *identifier* com o valor 0x00, passado a máquina de estados finitos para o estado *Transmit_ ACK*, esperando que a informação tenha sido enviada. Quando tal acontece, a FSM regressa ao estado *IDLE*.

Caso não seja necessário enviar um *acknowledge*, é testado se foi feito um pedido para envio de informação. Para verificar a existência de um pedido, é verificado o sinal *send*. Em caso afirmativo, são activados sinais para o módulo anterior, indicando que

uma nova transferência deve ser feita. Após isto, a máquina de estados finitos passa para o estado *Transmit_Packet,* o qual espera pela confirmação do envio do pacote. Quando tal acontece, o *timer* pseudo-aleatório começa a contar e a FSM regressa ao estado *IDLE,* repetindo-se este ciclo.

5.5.9 Módulo Transceiver

Este bloco engloba todos os blocos construídos até ao momento e através de uma máquina de estados finitos facilita a interface com o módulo protocolo para a transmissão de informação. A recepção não sofre qualquer processamento. O diagrama de blocos seguinte, mostra este componente. É de relembrar que o componente configurador possui mais sub-camadas.

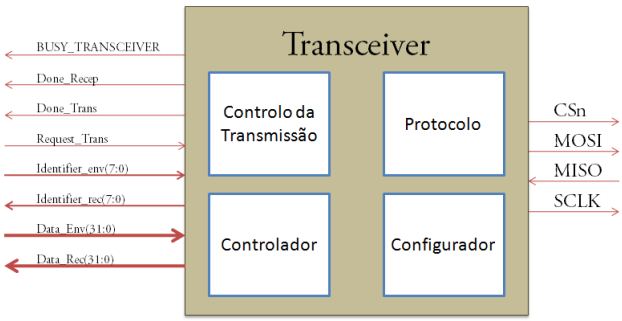

Figura 5.32: Módulo *Transceiver*

O respectivo diagrama de fluxo de sinal da FSM encontra-se representado na figura seguinte:

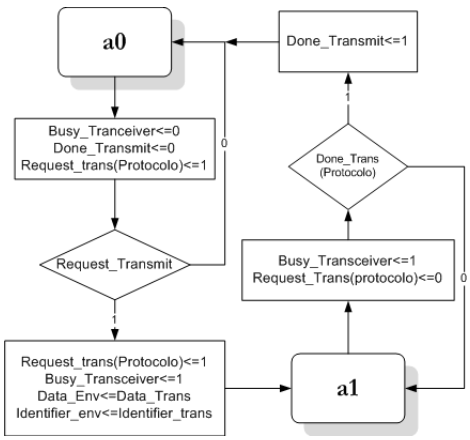

Figura 5.33: Diagrama de fluxo de sinal do bloco *Transceiver*

Esta máquina encontra-se, por defeito, no estado a0. Neste estado, todos os sinais que evocam um pedido ao componente protocolo encontram-se a zero. Neste,

é testado, igualmente, se foi feito algum pedido (sinal *Request_ Transmit*) por parte da camada superior. Em caso afirmativo, é feito então o pedido ao bloco "protocolo" e a saída *busy_ transceiver* toma o valor lógico '1', indicando à camada chamadora que o módulo se encontra ocupado. Após a transição para o estado a1, o sinal *Request_ Transmit* toma o valor lógico 0, uma vez que o pedido já se encontra a ser processado. Neste estado, é feita uma espera pela concretização da transferência, a qual é identificada pelo sinal *done_ trans* vindo do componente protocolo. Neste momento, a saída *done_ trasmit* toma o valor lógico '1', indicando que a informação foi enviada e que o módulo se encontra livre para uma nova transmissão. Para finalizar, a máquina volta ao estado inicial, repetindo-se todo o ciclo.

5.5.10 Módulo CC1101

Este componente encerra toda a interface com o módulo de rádio frequência e permite a criação de uma interface simples de utilização. O diagrama de blocos deste componente encontra-se representado na figura seguinte.

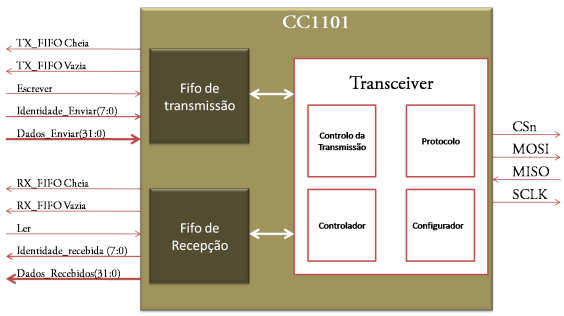

Figura 5.34: Módulo CC1101

Observando a figura anterior, é possível verificar que foram adicionadas duas FIFOs, encontrando-se estas ligadas ao componente *transceiver*. Estas são extremamente necessárias, uma vez que, sendo a taxa de transmissão do módulo wireless relativamente baixa em relação à velocidade do oscilador da placa de desenvolvimento, estas são capazes de armazenar informação que irá ser enviada, posteriormente, quando possível, não sendo por isso necessário à entidade que utiliza esta interface esperar pela conclusão da transmissão libertando desta forma recursos.

Para gerar estas FIFOs recorreu-se a um módulo IP designado por *Fifo Generator*. Este módulo permite definir vários parâmetros, tais como o tamanho da fifo pretendida, os sinais de controlo para o exterior, etc. Uma vez que cada pacote de informação possui 40 bits de informação (dados + identificador), foi definido neste módulo IP uma FIFO

com o tamanho de 1024x40 bits, sendo este um valor aceitável para a aplicação em causa.

A interface de comunicação com o módulo wireless tornou-se, deste modo, bastante simples, sendo necessários apenas 10 sinais de uso simples para efectuar uma transmissão ou recepção via comunicação wireless:

- **TX_FIFO_Cheia -** Este sinal indica quando a FIFO de transmissão se encontra cheia;
- **TX_FIFO_Vazia -** Este sinal indica quando a FIFO de transmissão se encontra vazia;
- **Escrever -** Sinal indicando que vai escrever informação na FIFO de transmissão;
- **Identidade_Enviar -** Possui o valor da identidade a enviar. (Relembrar que não pode ser usado o valor 0x00);
- **Dados_Enviar -** Informaçãoa transmitir.
- **RX_FIFO_Cheia -** Este sinal indica quando a FIFO de recepção se encontra cheia;
- **RX_FIFO_Vazia -** Este sinal indica quando a FIFO de recepção se encontra vazia;
- **Ler -** Sinal indicando que quer ler informação da FIFO de recepção;
- **Identidade_Recebida -** Possui o valor da identidade recebida;
- **Dados_Recebidos -** Informação recebida.

Escrita na FIFO de transmissão

Para efectuar uma escrita na FIFO, apenas é necessário colocar a informação relativa ao campo *Identifier* e dos dados nos respectivos sinais. De seguida, deve ser activado o sinal *escrever* devendo manter-se activo durante um ciclo de relógio.

Leitura da FIFO de recepção

A FIFO de recepção tem um funcionamento um pouco diferente. Para a leitura de informação deste, é necessário possuir um ciclo de espera, isto é, quando se pretende efectuar uma leitura é necessário activar o sinal *ler* durante um ciclo de relógio podendo ler a informação recebida apenas no próximo ciclo.

5.6 Ligação do módulo CC1101 ao restante sistema

Por fim, é necessário efectuar a ligação do módulo wireless ao restante sistema construído, ou seja, a ligação aos processadores. Esta ligação deverá permitir remotamente a configuração do *instruction set* do co-processador, bem como o carregamento de um novo programa escrito na linguagem *assembly* para o processador de uso geral, permitindo, desta forma, a execução de um programa completamente diferente.

Alteração do *instruction set* do co-processador

Para efectuar a alteração do *instruction set* do co-processador remotamente é necessário efectuar duas alterações: A memória ROM, que se encontra no programador da unidade de controlo, deverá passar a memória RAM. O mesmo acontece com a memória que se encontra no programador no fluxograma programável tendo esta que passar a memória RAM. Com estas alterações, foram acrescentados sinais a estes componentes, de forma a endereçarem a posição de memória que deve ser escrita. A figura seguinte, mostra as alterações efectuadas, incluindo os sinais adicionados:

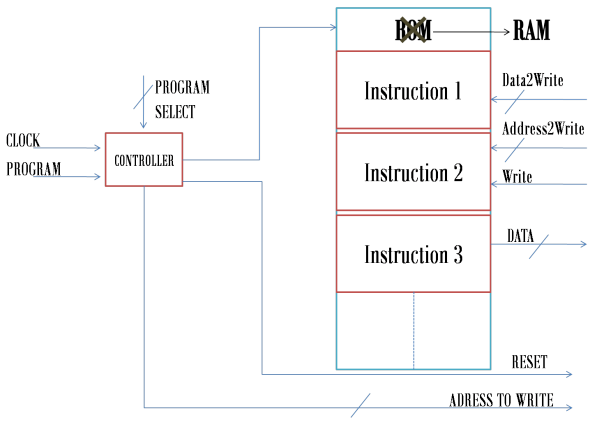

Figura 5.35: Programador da unidade de controlo

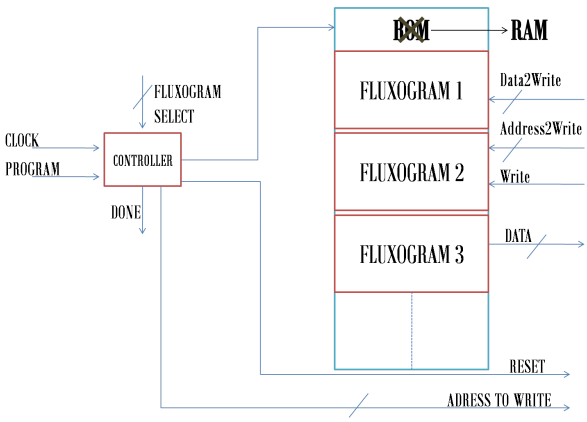

Figura 5.36: Programador do fluxograma

Reprogramação do programa *assembly*

Para efectuar o carregamento de um novo programa para o processador de uso geral é necessário efectuar uma alteração semelhante à necessária para alteração do *instruction set* do co-processador, isto é, a memória ROM que possuí o programa *assembly* deverá passar a memória RAM, sendo igualmente necessário a adição de sinais para a escrita na memória.

Diagrama final do sistema construído

O diagrama da figura seguinte mostra o sistema completo construído:

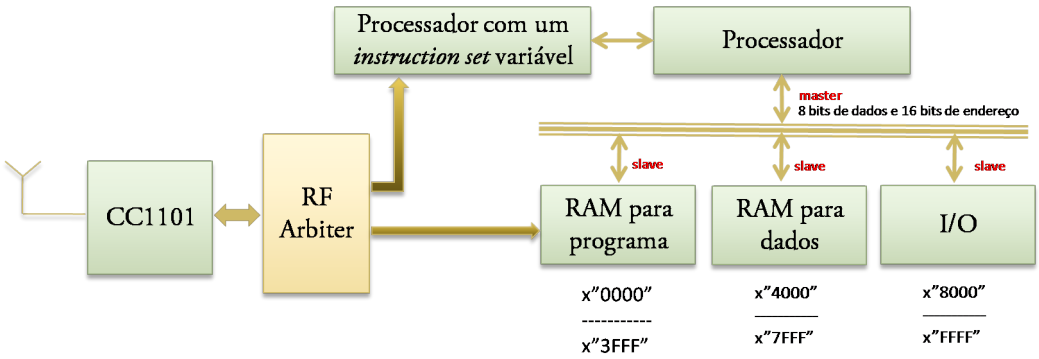

Figura 5.37: Diagrama de blocos do sistema completo construído

Observando a figura 5.37, é possível verificar a interligação entre os diversos blocos. É de notar a existência de um novo bloco, o qual foi designado por *RF Arbiter*. Este componente tem a função de ler a informação recebida por rádio frequência, procedendo à sua separação em conformidade com o identificador recebido, enviando de seguida a informação para o componente a que se destina. É através deste componente que é feita a reprogramação dos diversos componentes, sendo esta controlada por uma placa de desenvolvimento situada remotamente.

A figura seguinte, mostra a ligação deste componente ao restante sistema:

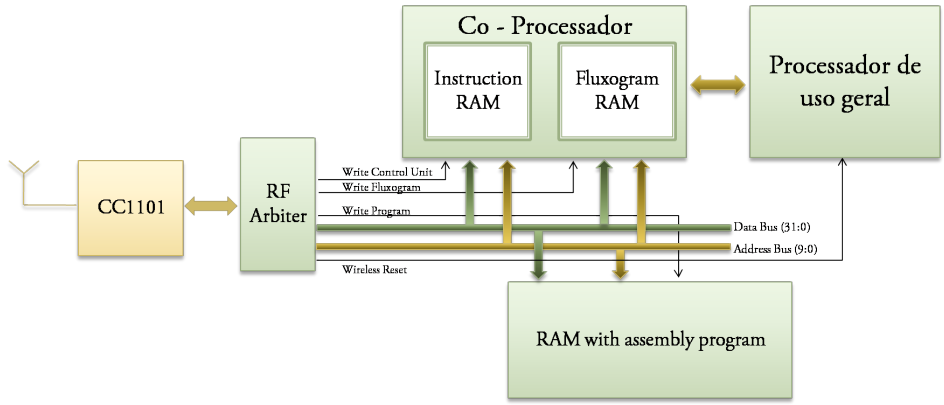

Figura 5.38: Sinais utilizados para a reprogramação remota

Analisando a figura anterior, é possível observar que este componente liga a todas as memórias RAM através de um barramento de dados e de endereços comum, sendo utilizados 3 sinais separados para indicar qual das memórias deve ser escrita com o endereço e dados que se encontram no barramento. Uma vez que é carregada nova

informação, torna-se necessário que o processador não se encontre a processar enquanto são carregados os novos dados. Deste modo, foi criado o sinal *wirelessReset,* permitindo fazer o *reset* ao processador de uso geral inactivando-o de qualquer operação. Quando este sinal é inactivado, o processador retoma o seu processo de funcionamento normal, mas correndo um novo programa se tal for carregado. Todos os sinais apresentados são controlados via rádio frequência, sendo o valor destes actualizados em conformidade com o identificador recebido. A tabela seguinte, mostra a correspondência entre o identificador e a alteração dos sinais que este desencadeia:

Identifier	Operação
0x01	ProgramRam(Address)<=DataReceive; WirelessReset<=1;
0x02	WirelessReset<=0;
0x03	Address<=Address+1; WirelessReset<=1;
0x04	ReprogrammableUnitControlRam(Address)<=DataReceive;
0x05	FluxogramRam(Address)<)DataReceive;
0x10	Address<=0;

Tabela 5.1: Correspondência entre o identificador e operação remota desencadeada

Programador Remoto

Visto um dos objectivos deste trabalho a configuração remota do sistema computacional construído é necessário implementar numa placa de desenvolvimento mecanismo de reprogramação com ligação ao módulo wireless.

Este sistema remoto deve ser então capaz de enviar um novo programa *assembly* bem como o novo set de instruções para o co-processador. Esta informação pode ser guardada em memória sendo apenas necessário quando da necessidade de reprogramação lê-la e enviando os correctos identificadores, conseguindo-se desta forma alterar o comportamento do sistema computacional remotamente. O esquema seguinte mostra o sistema completo construído:

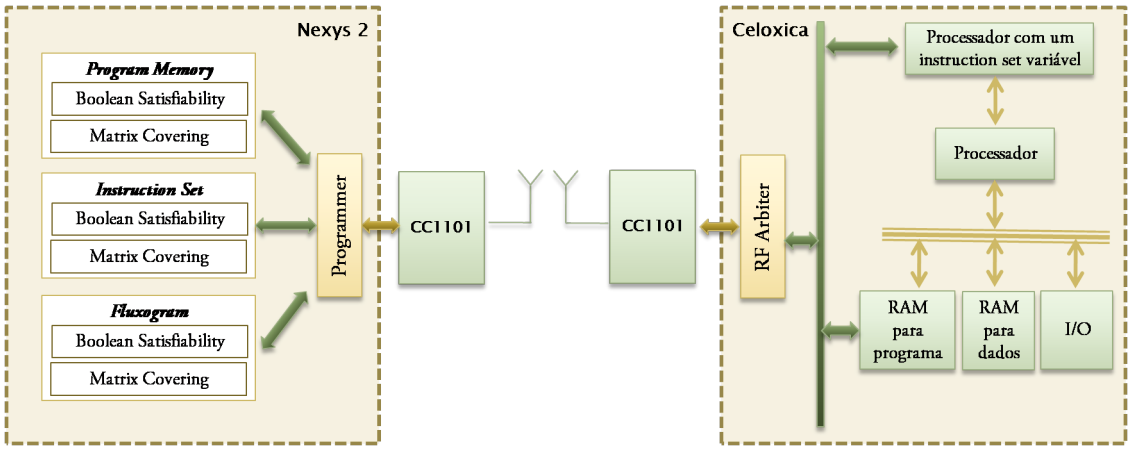

Figura 5.39: Interacçao remota

O módulo programador, em caso de pedido de reprogramação, segue os seguintes passos:

1. Envia um pacote com o identificador igual a 0x10, colocando remotamente o endereço a escrever nas memórias a zero;
2. É enviada informação do programa *assembly* com o respectivo identificador igual a 0x01;
3. É enviado um pacote com o identificador igual a 0x03 incrementando assim o endereço a escrever;
4. É lido endereço seguinte da memória que contêm o programa *assembly* e repetindo os passos 2,3 e 4 até que todo o programa assembly seja carregado;
5. É enviada informação sobre as instruções para o co-processador. Este processo é igual ao descrito para escrita na memória com o programa *assembly,* alterando-se

apenas o endereço de envio 0x01 para 0x04 e os dados enviados dizem respeito à memória que contém a informações sobre as instruções.

6. Por fim, é enviada a informação acerca do fluxograma para o co-processador. Este processo é igual ao anterior, sendo apenas alterado o endereço de envio para 0x05 e os dados enviados encontram-se na memória que contêm informações sobre o fluxograma reprogramável.

5.7 Resultados e Discussão

Nesta secção são apresentados os resultados. Visto a análise em relação ao sistema constituído pelos processadores já foi referida, nesta secção iremos discutir apenas os resultados de síntese obtidos para os componentes que compõe a stack protocolar. Serão também analisados alguns resultados práticos no que diz respeito às características do módulo wireless utilizado.

Utilização de recursos da interface sem fios desenvolvida

De seguida, são apresentados os resultados sobre a utilização de recursos da interface RF nas duas placas de desenvolvimento utilizadas:

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
CC1101	11/691	7/821	12/704	0/0	0/6	0/1	0/0	0/0
Transceiver_Module	45/606	87/733	4/652	0/0	0/0	0/1	0/0	0/0
Protocol_Control	131/137	164/172	137/144	0/0	0/0	1/1	0/0	0/0
Transmission_Control	88/88	123/123	54/54	0/0	0/0	0/0	0/0	0/0
Configurador	53/158	31/140	81/207	0/0	0/0	0/0	0/0	0/0
Controlador	174/178	203/211	237/243	0/0	0/0	0/0	0/0	0/0
RX_FIFO	0/53	0/54	0/31	0/0	0/3	0/0	0/0	0/0
TX_FIFO	0/21	0/27	0/9	0/0	0/3	0/0	0/0	0/0

Tabela 5.2: Utilização de recursos do componente CC1101 na placa de desenvolvimento Celoxica

Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM	MULT18X18	BUFG	DCM
CC1101	11/715	7/840	13/723	0/0	0/0	0/6	0/0	0/0
Transceiver_Module	45/598	87/725	4/648	0/0	0/0	0/0	0/0	0/0
Protocol_Control	127/132	156/164	132/139	0/0	0/0	0/0	0/0	0/0
Transmission_Control	89/89	123/123	54/54	0/0	0/0	0/0	0/0	0/0
Configurador	51/155	31/140	81/207	0/0	0/0	0/0	0/0	0/0
Controlador	173/177	203/211	238/244	0/0	0/0	0/0	0/0	0/0
RX_FIFO	0/53	0/54	0/31	0/0	0/0	0/3	0/0	0/0
TX_FIFO	0/53	0/54	0/31	0/0	0/0	0/3	0/0	0/0

Tabela 5.3: Utilização de recursos do componente CC1101 na placa de desenvolvimento Nexys2

A respectiva utilização de recursos em percentagem encontra-se representada na tabela seguinte.

Module	Slices	Slice Reg	LUTs
Celoxica	≈ 5.2%	≈ 3%	≈ 2.6%
Nexys 2	≈ 15%	≈ 9%	≈ 8%

Tabela 5.4: Utilização de recursos em relação à capacidade total de cada uma das placas de desenvolvimento

Observando os resultados, é possível verificar que os recursos utilizados por esta interface são relativamente baixos. No entanto, esta interface pode ainda ser alvo de melhoramentos, principalmente no que diz respeito à passagem das memórias utilizadas, as quais se encontram implementadas em memória distribuída, para Block RAMs.

Testes às características do módulo *wireless*

Por forma a avaliar alguns parâmetros do módulo wireless utilizados, foram realizados alguns testes. Estes, contemplam distâncias máximas, taxa de transferência, consumo entre outros.

Para efectuar estes, foi criado um projecto de teste, por forma a avaliar os parâmetros mais importantes. Este, envia constantemente um contador via rádio frequência e fornece algumas informações no monitor VGA, tais como: número de pacotes recebidos, número de pacotes reenviados, informação recebida, número de pacotes *acknowledge* entre outros. Este projecto foi então carregado para as duas placas de desenvolvimento.

Para a realização destes testes foram utilizados os seguintes parâmetros do *módulo wireless*:

- **Modulação:** GFSK;

- ***Datarate*:** 100KBaud;
- **Tamanho do pacote de dados fixo:** 6 Bytes;
- ***AutoFlush* CRC**;
- **Verificação de endereços;**
- **4 Bytes de palavra de sincronização.**
- **6 Bytes de *preamble*;**

Teste 1 - Módulos *Wireless* separados de uma distância de aproximadamente 1m com/sem FEC

Por forma a avaliar o comportamento da comunicação RF, em condições que podemos designar de ideais, foram carregados os projectos de teste para as placas de desenvolvimento, separando-se os módulos a uma distância de aproximadamente 1m. Neste teste foi activada/desactivada a opção de correcção automática de erros (FEC) para verificar o seu impacto na taxa de transferência. Obtiveram-se assim os seguintes resultados:

	S/ FEC		C/ FEC	
	Transceiver 1	Transceiver 2	Transceiver 1	Transceiver 2
Pacotes enviados	520,049	520,079	318,337	318,363
Último valor enviado	520,048	520,078	318,336	318,362
Pacotes recebidos	520,079	520,049	318,363	318,337
Último valor recebido	520,078	520,048	318,362	318,336
Acknowledges enviados	520,079	520,049	318,363	318,337
Reenvios	0	0	0	0
Taxa de transmissão	≈ 5.8 Kbit/s	≈ 5.8 Kbit/s	≈ 3.5 Kbit/s	≈ 3.5 Kbit/s
Taxa de transmissão incluindo acesso ao meio	≈ 74 Kbit/s		≈ 45 Kbit/s	
Taxa de transmissão incluindo *Acknowledges*	≈ 23.1 Kbit/s		≈ 14.1 Kbit/s	
Taxa de transferência	≈ 11.6 Kbit/s		≈ 7.1 Kbit/s	

Tabela 5.5: Resultados obtidos a uma distãncia de aproximadamente 1m, durante aproximadamente 1 hora

Observando os resultados, é possível verificar que nenhum pacote se perdeu, uma vez que, o número de reenvios possui um valor nulo, o que nos leva a concluir que realmente é possível considerar que para esta distância nos encontramos nas condições ideais. É possível verificar igualmente, que o número de pacotes recebidos pelos dois módulos é igual ao número de pacotes enviados pelo módulo oposto.

Em relação à taxa de transmissão, verifica-se que esta é semelhante nos dois módulos *wireless* o significa que a interface construída permite uma boa divisão da largura de

banda pelos dois módulos. A taxa de transmissão obtida foi cerca de 5.8 Kbits/s o que perfaz uma taxa de transferência de 11.6 Kbits/s (somatório das taxas de transmissão de cada um dos módulos). Contabilizando agora a restante informação para além da "útil", isto é, o envio dos pacotes *acknowledge* e dados de acesso ao meio (bits de sincronização e de preamble), obtendo-se uma taxa de transferência de 74 Kbits/s. No entanto, o módulo foi configurado para uma taxa de 100KBaud. Esta diferença de valores tem como uma das origens o facto da utilização de um tempo de espera, entre o envio de um pacote e o próximo. Uma vez que, existe um tempo subjacente à passagem do módulo do estado de transmissão para recepção e vice-versa, e como não é utilizado o tamanho máximo do pacote, para uma mesma quantidade de informação é necessário fazer mais transições entre estados, levando inevitavelmente à diminuição da taxa de transferência, constituindo assim outra causa da discrepância entre os valores teóricos e práticos da taxa de transmissão. Por fim, o valor teórico não contabiliza uma transmissão bidireccional, o que leva a uma diminuição da taxa de transferência.

Comparando agora os resultados obtidos com/sem correcção de erros, é possível verificar que a taxa de transmissão (incluindo acesso ao meio) diminuiu bastante. Este facto é explicado pelas etapas de processamento extra quando da activação desta funcionalidade.

Teste 2 - Módulos *Wireless* separados de uma distância de aproximadamente 240m com/sem FEC

De seguida, foi avaliado o comportamento dos módulos para distâncias maiores. Para tal, foram colocados a uma distância de aproximadamente 240m.

Obtiveram-se os seguintes resultados:

	S/ FEC		C/ FEC	
	Transceiver 1	Transceiver 2	Transceiver 1	Transceiver 2
Pacotes enviados	118,973	115,787	79,839	79,927
Último valor enviado	118,972	115,786	79,838	79,926
Pacotes recebidos	120,681	124,731	79,951	79,854
Último valor recebido	115,786	118,972	79,926	79,838
Acknowledges enviados	120,681	124,731	79,951	79,854
Reenvios	11,666	11,765	64	40
Taxa de transmissão	≈ 5.3 Kbit/s	≈ 5.2 Kbit/s	≈ 3.5 Kbit/s	≈ 3.6 Kbit/s
Taxa de transmissão incluindo acesso ao meio	≈ 68.3 Kbit/s		≈ 45.5 Kbit/s	
Taxa de transmissão incluindo *Acknowledges*	≈ 21.3 Kbit/s		≈ 14.2 Kbit/s	
Taxa de transferência	≈ 10.5 Kbit/s		≈ 7.1 Kbit/s	

Tabela 5.6: Resultados obtidos a uma distãncia de aproximadamente 240m, durante aproximadamente 15 min

Observando os resultados, é possível verificar imediatamente que existem pacotes que se perderam, uma vez que, existe pacotes reenviados. Este facto, leva a uma inevitável queda na taxa de transmissão. No entanto, apesar de pacotes perdidos, devido ao protocolo implementado, não existe informação perdida, indicando deste modo, que a interface criada é viável. Após o estudo sem a utilização do corrector automático de erros, este foi accionado. Verificou-se como seria de esperar uma melhoria nos resultados no que diz respeito ao número de pacotes reenviados, sendo este valor muito baixo. Verifica-se como no teste anterior que a taxa de transmissão com esta funcionalidade diminui, pelas razões já explicadas. No entanto, esta taxa de transmissão manteve-se aproximadamente igual ao teste anterior ao contrário da taxa de transmissão sem correcção automática de erros. Este facto, leva-nos a concluir que existe um ponto em que a taxa de transmissão sem/com corrector de erros irá ser praticamente igual, tornando-se a partir deste viável a utilização desta funcionalidade.

Distância máxima

Apesar da distância considerada no teste 2, através das experiências efectuadas o módulo consegue receber até a uma distância de aproximadamente 270m. No entanto, para este valor existe já uma grande quantidade de pacotes a serem reenviados, tornando a ligação sem fios muito lenta. Os testes realizados foram feitos sem obstáculos, pelo que estes valores apenas são válidos para este tipo de ambiente. Através das experiências feitas, foi observado igualmente o fenómeno conhecido por **desobstrução de Fresnel** isto é, foi verificado que para longas distâncias é necessário que os módulos se encontrem a uma altura relativamente elevada para que a transmissão seja possível (para mais informações consultar [56]). Por todas estas razões, foi considerada como distância máxima 240m.

Consumo

Visto uma das grandes preocupações na escolha do módulo *wireless* era o consumo energético, foi medido o seu consumo energético à potência (programável) máxima e mínima. Obteve-se então para uma potência máxima (10dBm), um consumo de aproximadamente 9.8mA e 6.4mA para a potência mínima (-30dBm), sendo estes valores bastante baixos, tendo principalmente em conta a distância que atingem, conseguindo-se desta maneira uma boa relação potência - distância.

Podemos então concluir que todos os objectivos no que toca à interacção remota foram concluídos, conseguindo-se uma ligação fiável, de baixo custo, taxas de transferência suficientes e baixo consumo.

Interface gráfica

Tal como foi dito, foram elaborados algoritmos combinatórios para demonstração do sistema construído. Este, encontra-se verificado e correctamente a funcionar. A interface gráfica para demonstração encontra-se representada na figura seguinte. Esta diz respeito à placa de desenvolvimento que contém o processador com um conjunto de instruções variável.

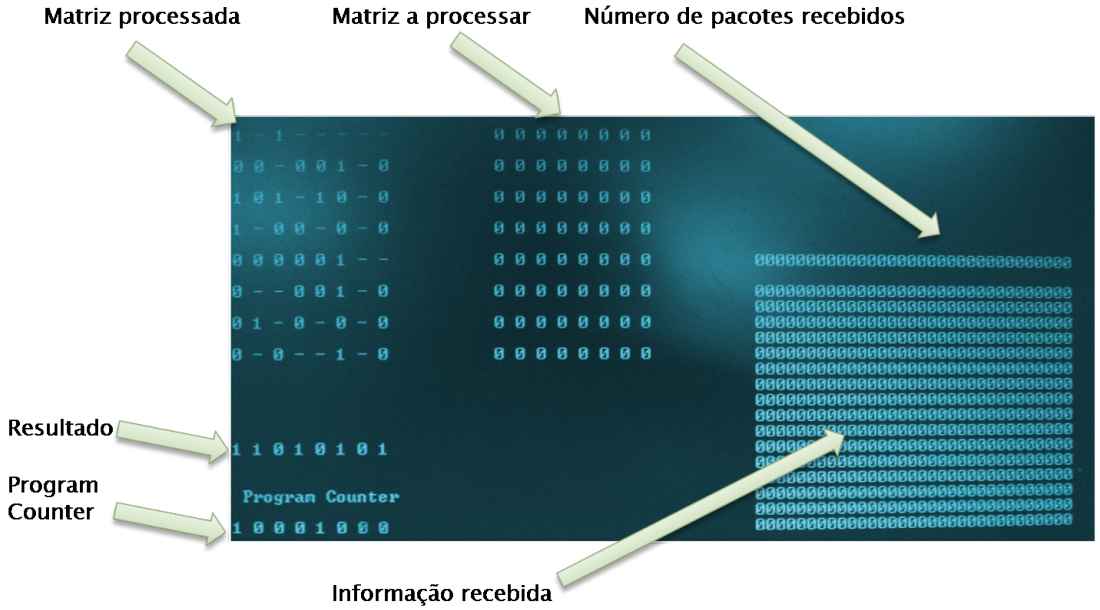

Figura 5.40: Interface do demonstrador

Capítulo 6

Conclusão e Trabalho Futuro

6.1 Sumário

Este capítulo completa a dissertação. Aqui é feito um breve resumo do trabalho realizado e principais conclusões. Para terminar são apresentadas algumas propostas para trabalho futuro.

6.2 Conclusão

Devido aos avanços tecnológicos a nível de sistemas computacionais dos quais se destacam os sistemas reconfiguráveis, existe cada vez mais uma tendência para a adopção deste tipo de dispositivos para aplicações na grande parte das áreas, quer de investigação quer comerciais. O grande impulsionador deste tipo de sistemas foi a FPGA, o que permitiu a construção de sistemas de grande complexidade com um reduzido tempo de projecto e uma grande flexibilidade no design, que continua a aumentar com a constante evolução na área da microelectrónica, bem como o constante aparecimento de novas ferramentas CAD para implementação, síntese e simulação para sistemas digitais.

Neste âmbito, é de grande interesse o desenvolvimento de novos sistemas para este tipo de dispositivos, o que, no contexto desta dissertação, foi a incorporação de um processador com um *set* de instruções variável remotamente, o que permitiu o desenvolvimento de uma nova forma de estruturação da arquitectura de computadores.

Numa primeira parte deste trabalho, foi efectuado um estudo sobre os diferentes tipos de sistemas computacionais e a organização de um computador. Foi possível analisar os conceitos fundamentais da execução de uma instrução num microprocessador.

Tentando tirar partido das grandes capacidades das FPGAs, foi então idealizada uma forma de reduzir o *instruction set* de um processador apenas ao estritamente

necessário para a execução de um determinado programa, o que levou ao estudo mais aprofundado da unidade de controlo de um microprocessador, assim como a utilização de máquinas de estados finitos reprogramáveis para a sua implementação. Este facto, permitiu a construção de uma unidade de controlo reprogramável, adaptando-se em conformidade com a instrução a executar. De forma a desenvolver o que designamos de processador, foi criado um *datapath* com especialização para análise de vectores binários e ternários controlados pela unidade de controlo reprogramável. Através da utilização da estrutura desenvolvida, foi possível concluir as previsões para uma das grandes vantagens deste tipo de arquitectura: a redução do espaço utilizado pela unidade de controlo. No entanto, foi igualmente possível verificar a grande desvantagem deste tipo de arquitectura: tempo de latência, isto é, existe sempre um tempo necessário entre o pedido e execução da instrução, uma vez que, a unidade de controlo tem de ser programada para executar a instrução em causa.

De forma a demonstrar o funcionamento, o processador desenvolvido foi acoplado a um processador de uso geral, desempenhando assim funções de co-processamento. O processador de uso geral utilizado foi, no entanto, modificado de forma a ser possível a construção de algoritmos de pesquisa combinatória, uma vez que estes necessitam de instruções para análise matricial e operações de recuo e avanço de maneira a encontrar a melhor solução.

Actualmente, devido à grande afluência às tecnologias sem fios, é de grande interesse a utilização de sistemas rádio para a realização de várias tarefas remotamente. Neste trabalho, pretendeu-se adicionar mais uma função: a reprogramação de um *instruction set* remotamente, bem como alteração do programa que um processador executa. Apesar da grande utilização de tecnologias sem fios, continua a verificar-se a falta de componentes nas livrarias IP fornecidas pelos fabricantes de FPGA, o que motivou ainda mais a utilização de uma interface sem fios para interacção remota de FPGAs.

Devido à grande preocupação ambiental, houve um grande interesse na escolha de um módulo wireless de baixo consumo, pelo que foi utilizado o transceiver CC1101 do fabricante Texas Instruments. Este, provou ser uma boa escolha demonstrando uma boa relação a nível de desempenho-preço, bem como a capacidade de utilização a grandes distâncias e baixo consumo.

Numa segunda parte, foi então desenvolvida toda a interface para a comunicação sem fios entre FPGA, o qual englobou a construção de circuitos impressos para ligação às mesmas. Para a construção desta, foi necessário o desenvolvimento de várias camadas, começando pelo protocolo de comunicação com o módulo wireless (SPI) até à simples interface com o projectista através de FIFO. De forma a dar uma percepção ao *designer* da contínua recepção e transmissão foram feitas camadas intercalares,

automatizando o processo de configuração do transceiver para recepção e transmissão. De forma a melhorar a qualidade da ligação remota, foi igualmente implementado um protocolo de comunicação que provou uma grande imunidade à perda de pacotes sendo este protocolo implementado directamente na FPGA. Criou-se, desta forma, um componente de fácil utilização podendo ser reutilizado em muitas outras aplicações.

Para finalizar, foi interligado o sistema constituído pelos dois processadores à interface sem fios que permitiu a reprogramação do *instruction set* remotamente, bem como a alteração do programa a executar pelo processador. Para tal, foi necessária a construção de outro sistema enviando e controlando a informação necessária à reprogramação. A este sistema desenvolvido igualmente em FPGA, designou-se por Programador. De forma a demonstrar os resultados, estes foram projectados no monitor VGA, permitindo uma fácil execução dos algoritmos combinatórios (Satisfação booleana e Cobertura de Matrizes) desenvolvidos, bem como a percepção da reprogramação remotamente.

É de salientar que todos os componentes apresentados neste trabalho foram implementados e verificados, funcionando todos eles correctamente.

Podemos então concluir que todos os objectivos propostos neste trabalho foram concluídos, conseguindo-se, desta forma, demonstrar as capacidades dos dispositivos lógicos programáveis para construção de microprocessadores quer especializados quer de uso geral e a sua flexibilidade para a construção de stack protocolares para interacção remota.

6.3 Trabalho Futuro

Com o desenvolvimento deste trabalho resultaram algumas ideias para trabalho futuro destacando-se as seguintes:

Optimização do processador desenvolvido

Tal como foi visto, é possível reduzir ainda mais o tamanho do processador com um *instruction set* variável. Isso permitirá uma maior adesão a este tipo de estrutura, principalmente para sistemas que necessitem de uma grande capacidade de periféricos, libertando, deste modo, o espaço ocupado pelo processador. Para além da redução de espaço, é necessário ainda encontrar formas mais eficientes da reprogramação da unidade de controlo, de forma a tornar o sistema o mais rápido possível.

Construção de um processador de uso geral com um *instruction set* variável remotamente

Uma vez que as bases para a construção de um processador com um *instruction set* variável remotamente se encontram desenvolvidas, é possível pensar, agora, no desenvolvimento de um processador de uso geral com estas características, motivando a uma maior utilização deste tipo de arquitectura.

O estudo efectuado, pode ser estendido, igualmente, a arquitecturas multi-core, conseguindo-se desta forma uma maior taxa de processamento.

Teste e optimização da interface sem fios

Apesar de terem sido aproveitadas uma grande quantidade de funcionalidades do transceiver CC1101, existem características deste módulo úteis não utilizadas. Neste contexto, é de grande interesse o estudo da melhor forma para as incluir na interface construída. Uma das características mais importantes não utilizadas, consiste na utilização de funções de poupança de energia, permitindo uma redução de potência do módulo quando da inexistência de necessidade de comunicação. Outra grande capacidade não aproveitada consiste na alteração das características rádio do módulo dinamicamente, isto é, a possibilidade de configurar o *transceiver* para receber, por exemplo, noutra modelação, taxa de transmissão entre outras características conforme as necessidades correntes.

Existe, igualmente, alguns testes que deverão ser feitos e, em caso de falha, corrigidos. Um desses exemplos consiste na ligação de múltiplos *transceivers*, recorrendo para isso a funcionalidade de detecção de endereços do módulo e efectuar testes à imunidade do sistema construído, uma vez que este apenas foi testado para uma ligação ponto-a-ponto.

Criação de uma rede sem fios para processamento paralelo em FPGAs

No contexto de aproveitamento do módulo *wireless* é possível, como projecto futuro, propor um sistema para processamento paralelo via rádio frequência, isto é, dividir um determinado problema por diversas FPGAs numa rede e, através de um sistema central de controlo, executar um algoritmo de grande complexidade, recebendo estes os resultados provenientes dos diversos módulos. Nesta rede podemos incluir processadores com *set* de instruções variável o que permite uma optimização de recursos numa rede de processamento paralelo. O conceito de reprogramação do programa a executar em cada uma das FPGAs é também aplicado.

Reconfiguração remota para aplicações na área do *cognitive radio*.

Foi concluído que as FPGAs continuam a impulsionar os sistemas de rádio cognitivo. Desta forma é possível aproveitar as suas características, de forma a tornar este conceito ainda mais emergente no quotidiano. Uma das aplicações futuras consiste na utilização de rádios que incorporam FPGAs, permitindo que estas sejam reconfiguradas remotamente para alteração das características de rádio a receber/transmitir. Umas das possíveis ideias, consiste em ter uma estação base que constantemente sente o espectro envolvente e, em conformidade com o que detecta, possuir a responsabilidade de reconfigurar os rádios cognitivos da sua rede, conseguindo-se, desta forma, facilitar a transição para uma nova frequência de transmissão de informação, o que leva inevitavelmente a um melhor aproveitamento do espectro.

Apêndice A

Instruções do Co-Processador e Processador de Uso geral

A.1 Sumário

Este apêndice descreve as diversas instruções do Co-Processador e Processador de uso geral. Inicia-se com uma explicação detalhada do fluxo da unidade de controlo para a execução de cada uma das instruções desenvolvidas. Para finalizar são apresentadas as instruções que compõem o processador de uso geral utilizado.

A.2 Instruções do Co-Processador

O Co-Processador desenvolvido pretende mostrar o funcionamento de uma arquitectura com uma unidade de controlo variável. O processador com um *instruction set* variável desenvolvido encontra-se representado na figura seguinte. A sua unidade de controlo bem como todos sinais de controlo encontram-se explicados no capítulo 4.

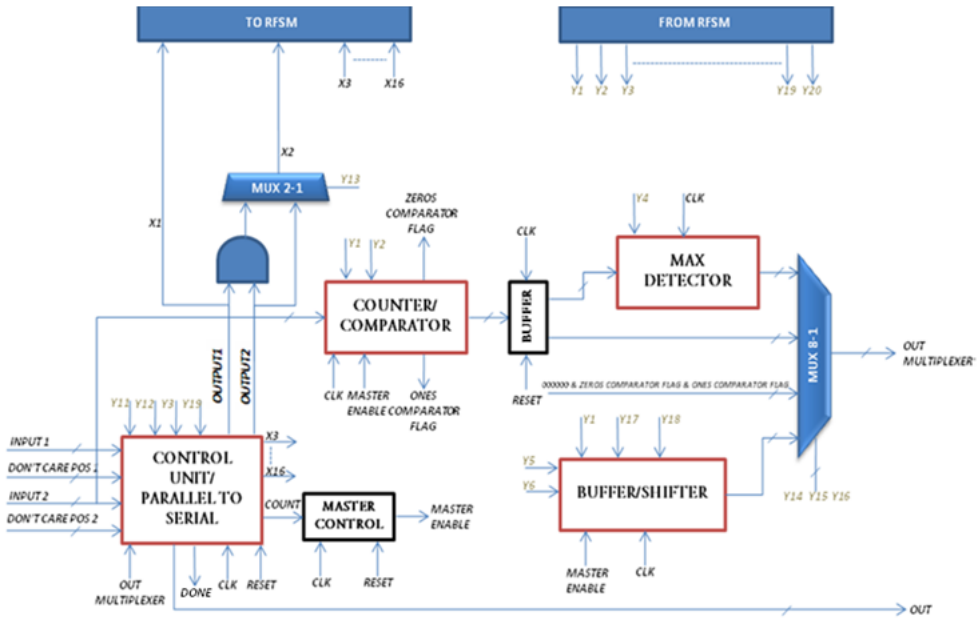

Figura A.1: Processador combinatório com um *instruction set* variável

As seguintes instruções foram desenvolvidas:

- Contar o número de uns de um vector binário;
- Contar o número de uns de um vector ternário;
- Contar o número de zeros de um vector ternário;
- Contar o número máximo consecutivo de uns de um vector binário;
- Contar o número máximo consecutivo de uns de um vector ternário;
- Contar o número máximo consecutivo de zeros de um vector ternário;
- Verificar se um vector ternário apenas contém valores *dont care*;
- Verificar se um vector contém apenas uns/zeros;
- Verificar se um vector não contém apenas uns/zeros;
- Verificar se um vector contém K elementos a um/zero;
- Verificar a expressão *vectori and vectorj = vectorj;*
- Executar a operação vectori or vectorj;
- Verificar se dois vectores ternários são ortogonais;

Em todos os diagramas de fluxo de sinal que se seguem, estes não contemplam um estado que indica o final da execução de uma instrução. Isto deve-se ao facto, que o final da execução depende do componente *Master Control*, podendo por isso acabar em qualquer estado.

Contar o número de uns de um vector binário

O diagrama de fluxo de sinal correspondente a esta operação encontra-se representado na figura seguinte. Neste são representadas as saídas que se encontram activas em cada estado bem como o nome do estado.

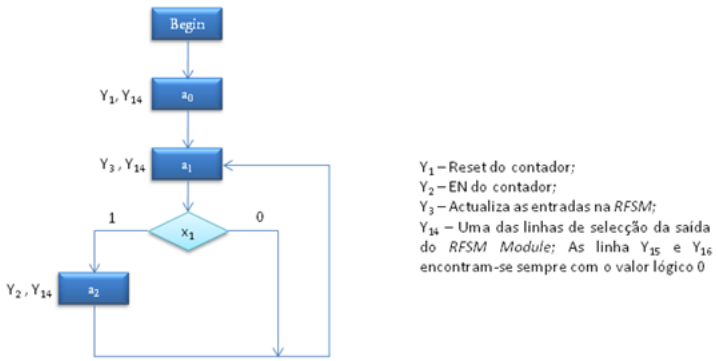

Figura A.2: Diagrama de fluxo de sinal para a instrução que permite contar o número de uns de vector binário

Tendo em vista contar o número de uns de um vector binário, a RFSM começa por activar a saída Y1 colocando assim o contador com o valor a 0. De seguida, esta transita para o estado a1 onde é activada a saída Y3. Esta irá pedir ao bloco *Control Unit/Parallel to Serial* para actualizar as entradas na RFSM para que o sinal de entrada comece a ser analisado. Neste ponto é analisada a entrada 1. Se o valor de entrada for igual a 1, a máquina irá transitar para o estado a2, caso contrário irá transitar automaticamente para o estado a1 e o ciclo repete-se. Caso transite para o estado a2, será activada a saída Y2 incrementando desta maneira o contador. Tal como foi visto o Multiplexer de saída permite seleccionar a saída de do processador combinatório. Para que a saída tome os valores que provêm do buffer (ver Figura A.1) é necessário que: Y14 = 1, Y15 = 0 e Y16 = 0.

Contar o número de uns de um vector ternário

Esta operação permite contar o número de uns de um vector ternário ou seja com valores lógicos '1', '0' e *don't care*.

O diagrama de fluxo de sinal correspondente encontra-se na figura seguinte:

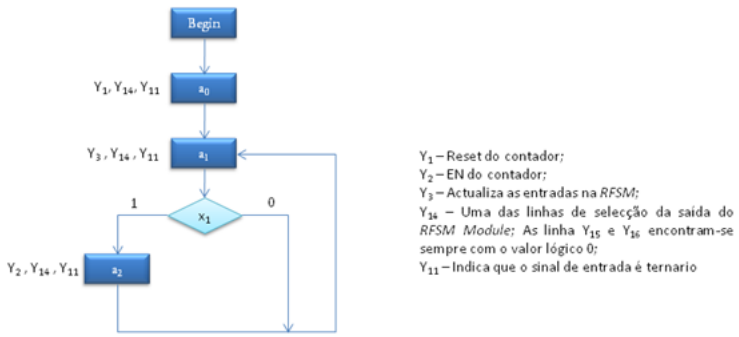

Figura A.3: Diagrama de fluxo de sinal para a instrução que permite contar o número de uns de vector ternário

Observando a figura anterior verifica-se que é em tudo semelhante ao anterior. A única diferença é que neste caso o sinal Y11 se encontra com o valor '1'. Isto indica ao componente *Control Unit/Parallel to Serial* que o vector de entrada deverá ser considerado como um vector ternário, pelo que este deverá analisar o vector que contêm a posição dos *don't cares* dos vectores de entrada. Como o sinal Y12 se encontra a zero, quando é encontrado um don't care o valor que será enviado para a RFSM terá o valor lógico '0'. Deste modo a memória que contêm informação sobre esta operação e a anterior será semelhante mudando apenas a memória de saída contendo o valor correspondente da saída Y11 com o valor lógico a '1'.

Contar o número de zeros de um vector ternário

Esta operação é semelhante à anterior, no entanto ao invés de contar o número de uns, conta o número de zeros de um vector ternário. O seu diagrama encontra-se representado na Figura A.3.

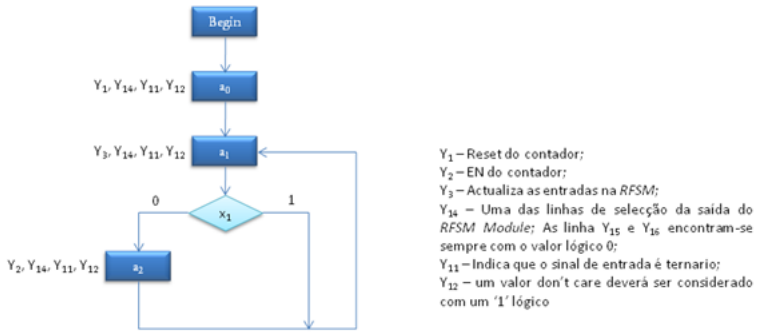

Figura A.4: Diagrama de fluxo de sinal para a instrução que permite contar o número de zeros de vector ternário

Observando a figura anterior, verifica-se a sua semelhança com a Figura A.3. Neste caso, como queremos analisar o número de zeros, a máquina de estados reprogramável deverá transitar para o estado a2 não quando encontrar um '1' lógico mas sim um '0'. Deste modo, iremos contar não o número de uns mas sim o número de zeros. Esta máquina, para funcionar exige que outra saída seja alterada: Y12. Esta quando com o valor lógico '1' irá indicar ao componente *Control Unit/Parallel to Serial* que deverá considerar o valor lógico *don't care* como sendo '1'. Deste modo os *don't cares* não serão contabilizados na contagem.

Encontrar o número máximo de uns consecutivos de um vector binário

Esta operação permite a contagem do número máximo de uns consecutivos de um vector binário. Um possível diagrama de fluxo de sinal para esta operação encontra-se representado na figura seguinte:

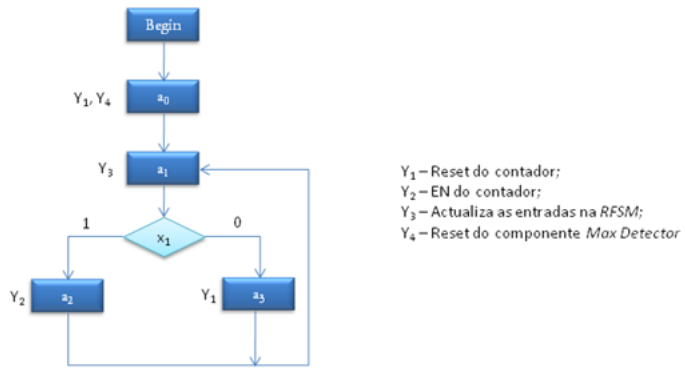

Figura A.5: Diagrama de fluxo de sinal para encontrar o número máximo de uns consecutivos de um vector binário

Nesta operação foi utilizado um novo componente do datapath: o *Max Detector*. A máquina de estados finitos começar por colocar o contador a zero, bem como o valor de saída do bloco *Max Detector*. Estas operações são executadas no estado a0. De seguida é actualizado os valores das entradas na RFSM. Se o valor de entrada possui o valor lógico '1' então o contador irá incrementar de uma unidade. Caso contrário irá ser reiniciado. Todo este processo se irá repetir até que o sinal *Master Enable* tome o valor '0'. Observando a Figura 6 verifica-se que o valor do contador entra no bloco *Max Detector* pelo que será detectado o número máximo de uns consecutivos de um vector binário. A nossa saída será então o resultado devolvido pelo bloco *Max Detector*. Para este ser seleccionado é necessário que todas as entradas de selecção do multiplexer de saída se encontrem a zero (Y14 = Y15 = Y16 = 0).

Encontrar o número máximo de uns consecutivos de um vector ternário

Esta operação é em tudo idêntica à anterior. A única diferença é que temos de indicar que o vector deve ser interpretado como vector ternário. Isto é feito tal como foi dito anteriormente pela activação da saída Y11. Para a contagem do número de uns é conveniente que o valor lógico *don't care* seja interpretado como valor lógico '0'. Assim a saída Y12 terá de possui o valor '0'. O diagrama de fluxo de sinal desta operação encontra-se representado na Figura A.6.

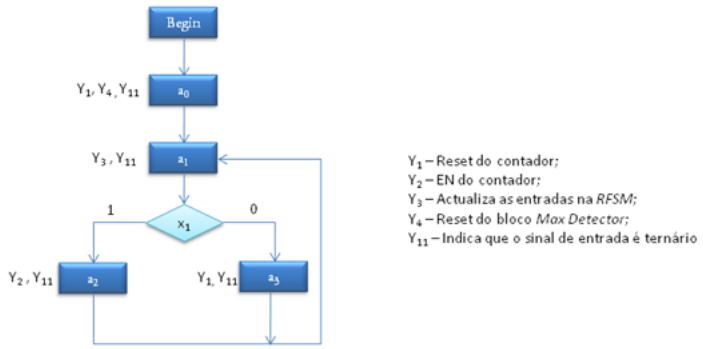

Figura A.6: Diagrama de fluxo de sinal para a instrução que permite contar o número máximo de uns consecutivos de um vector ternário

Encontrar o número máximo de zeros consecutivos de um vector ternário

A operação seguinte permite contar o número máximo de zeros consecutivos de um vector ternário. O seu diagrama de estados encontra-se representado na Figura A.7.

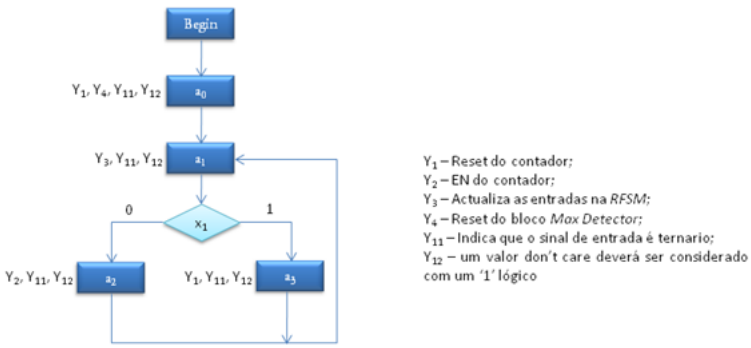

Figura A.7: Diagrama de fluxo de sinal para a instrução que permite contar o número máximo de zeros consecutivos de um vector ternário

Observando a figura anterior facilmente se verifica a sua semelhança com a Figura A.6. Neste caso no entanto, a máquina de estados transita para o estado a3 quando a entrada possuí o valor lógico '0' e não '1', conseguindo-se deste modo contar o número de zeros invés do número de uns. É de notar igualmente a utilização do bloco *Max Detector* tal como na operação anterior.

Verificar se um vector contém só uns (zeros)

A operação que se segue tem como objectivo detectar se um vector binário contém todos os seus valores com o nível lógico '1' ou com o nível lógico '0'. O seu diagrama de estados encontra-se representado na figura seguinte.

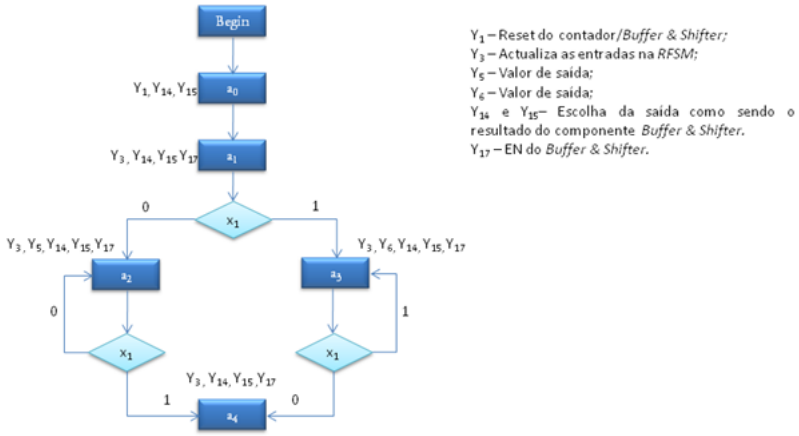

Figura A.8: Diagrama de fluxo de sinal para a instrução que permite verificar se um vector contém só uns (zeros)

A RFSM começa por colocar a saída do componente *Buffer/Shifter* a zero. Tal é feito a partir do sinal Y1. Como a saída desta operação será o resultado deste componente é necessário que escolher esta no multiplexer de saída. Para tal basta colocar os sinais Y14 e Y15 com o valor lógico '1'. De seguida é actualizada a entrada da RFSM colocando activando a saída Y3. De forma a colocar o componente *Buffer/Shifter* em funcionamento é activado igualmente o sinal Y17. Como se pretende que a saída seja o valor de "000000"&Y6&Y5 o sinal Y18 mantém o valor lógico '0'. De seguida é então analisada a entrada. Se esta for igual a zero a máquina de estados finitos irá transitar para o estado a2, caso contrário transita par ao estado a3. Caso o primeiro valor da entrada se encontre a zero, a saída Y5 irá ser activada. É então pedido a actualização da entrada da RFSM e são analisados os próximos valores da entrada. Se todos valores analisados se mantiverem com o valor '0' então a máquina irá permanecer no estado a2, obtendo-se assim a saída "00000001". O mesmo acontece caso o primeiro valor analisado se encontre a '1'. Neste caso a máquina transita para o estado a3 e se nenhum '0' aparecer esta irá permanecer neste estado obtendo-se na saída "00000010". Estando a máquina quer no estado a2 quer no estado a3 se o valor de entrada variar, esta irá transitar para o estado a4 permanecendo neste até a operação terminar. A saída neste caso será um vector com todos os valores a zero indicando que nenhuma das condições foi verificada.

Verificar se um vector não tem uns (zeros)

Esta operação é muito semelhante à anterior se tivermos em conta que é a sua negação. Isto é, um vector não possuirá uns lógicos se o vector estiver preenchido totalmente com zeros. O mesmo acontece na verificação se um vector genérico não tem zeros, isto é, apenas possui uns. Deste modo a única alteração que deverá ser feita relativamente ao diagrama anterior é colocar a saída Y5 activa invés da saída Y6 quando a máquina se encontra no estado a3 e trocar a saída Y5 por Y6 no estado a2. Deste modo quando o vector a analisar não contêm uns a saída será "00000010" e quando não possui zeros a saída tomará o valor "00000001". Caso o vector possua zeros e uns a saída tomará o valor "00000000" como na operação anterior. O diagrama de fluxo de sinal desta instrução encontra-se representado na figura seguinte.

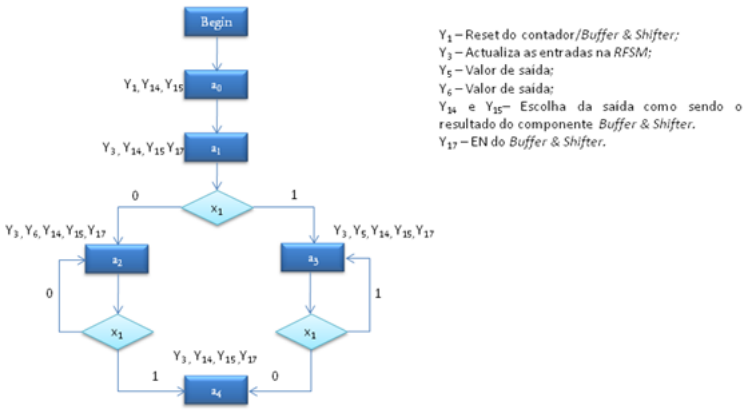

Figura A.9: Diagrama de fluxo de sinal para a instrução que permite verificar se um vector não tem uns (zeros)

Verificar se dois vectores ternários são ortogonais

A operação que se segue tem como função determinar se dois vectores ternários são ortogonais um em relação ao outro. Vamos começar por definir quando dois vectores ternários são ortogonais: Dois vectores ternários são ortogonais quando existe pelo menos um '1' e um '0' na mesma posição mas de diferentes vectores. Vejamos um exemplo:

Vector 1: 1-0-1110
Vector 2: 0-011111

Figura A.10: Ortogonalidade entre vectores

Caso seja encontrado um *don't care* em qualquer uns dos vectores numa determinada posição deste, automaticamente pode-se passar para a posição do vector seguinte. No algoritmo implementado os *don't cares* são tratados como valores lógicos '0'. Caso um *don't care* apareça num dos vectores automaticamente em ambos os vectores, na posição em causa será colocado um zero, conseguindo-se desta forma que esta posição não afecte o cálculo da ortogonalidade. O diagrama de fluxo de sinal desta operação encontra-se representado na figura seguinte:

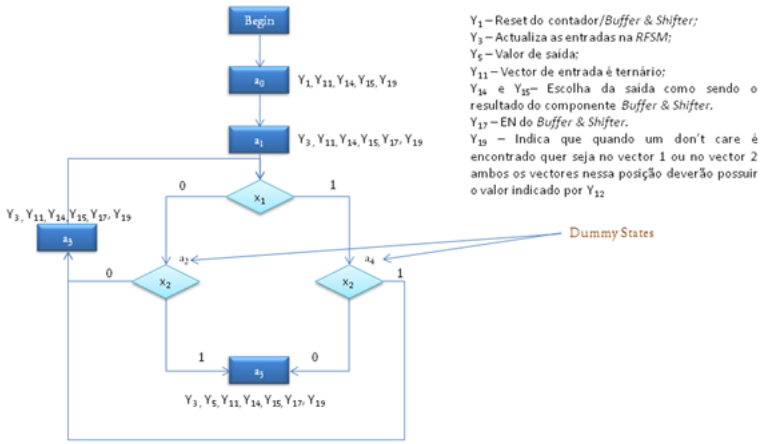

Figura A.11: Diagrama de fluxo de sinal para a instrução que permite verificar se dois vectores ternários são ortogonais

Analisando a Figura A.11 verifica-se que a máquina começa por fazer um *Reset* através da saída Y1 ao bloco *Buffer/Shifter*. É de salientar igualmente a activação da saída Y19 pela razão já explicada. Dá-se inicio então à verificação. A máquina testa os dois vectores num ciclo de relógio. Caso encontre um par de valores (1,0) ou (0,1) verifica-se a condição de ortogonalidade, transitando assim para o estado a5 e colocando a saída Y5 activa indicando deste modo que os vectores são ortogonais. A saída deste caso será então o resultado devolvido pelo bloco *Buffer/Shifter* e retornará o valor "00000001" em caso afirmativo. Caso contrário um vector nulo.

Verificar se um vector contém exactamente K componentes a um (zeros)

A operação seguinte permite verificar se um determinado vector possui K componentes com o valor lógico '1' ou '0'.

O valor K é introduzido na entrada Input 2 (ver Figura A.1) e o vector a analisar na entrada Input 1(ver Figura A.1). Relembrando a Figura A.1 observa-se que uma

das saídas disponíveis do bloco *Counter/Comparator* indica quando o valor do vector Input 2 é igual ao valor do contador (Ones Comparator Flag) bem com a subtracção do tamanho genérico dos vectores pelo valor armazenado em Input 2 (Zeros Comparator Flag). Estas *flags* encontram-se disponíveis para saída. Esta é escolhida activando a linha Y15. Em relação ao diagrama de fluxo de sinal, este é exactamente igual ao da operação contar o número de uns de um vector binário exceptuado a saída do multiplexer de saída que é diferemte. Deste modo a saída será "00000001" quando existe K elementos com o valor '1', "00000010" quando existem K elementos com o valor '0', "00000011" quando existem K elementos a '1' e a '0' e "00000000" quando a condição não é satisfeita.

Verificar se a expressão vectori and vectorj = vectorj está correcta

Esta operação permite verificar se a expressão vectori & vectorj = vectorj se encontra correcta. Para executar esta, é necessário primeiro efectuar a operação and. Para tal é necessário activar a saída Y13 para que a segunda entrada da RFSM seja o resultado da operação vectori & vectorj. Depois de executada a operação and é necessário fazer uma operação de comparação entre o resultado da operação and e o vectorj. É de notar que para a operação esteja correcta o vectorj deverá ser a entrada Input 1 e o vectori a entrada Input 2. O diagrama de fluxo de sinal desta operação encontra-se representado na figura seguinte:

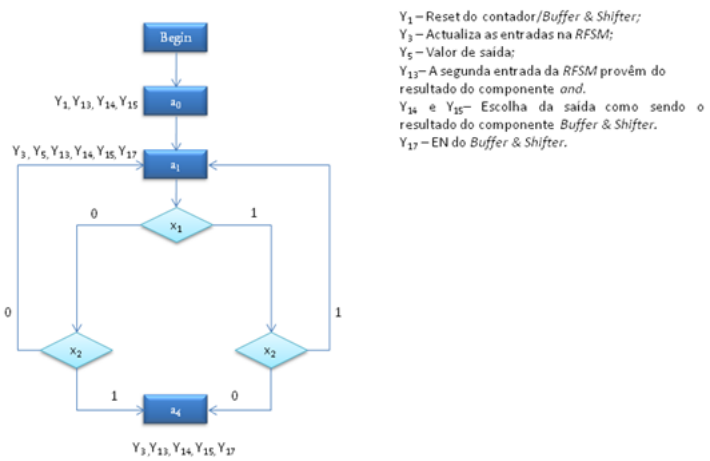

Figura A.12: Diagrama de fluxo de sinal para a instrução que permite verificar se a expressão vectori and vectorj = vectorj está correcta

Nesta operação o resultado provém do componente *Buffer/Shifter*. Tal como em

operações anteriores inicia-se a operação fazendo um reset ao bloco *Buffer/Shifter*. De seguida a máquina transita para o estado a1. Inicialmente considera-se que a expressão se encontra correcta, activando-se desta maneira a saída Y5. São analisadas então em cada ciclo de relógio um par de entradas. A maquina irá manter-se no estado a1 até que seja encontrado um par (1,0) ou (0,1) indicando desta forma que a expressão se encontra incorrecta. Quando tal acontece a máquina transita para o estado a4 mantendo-se neste até que a execução termine. Neste estado a saída Y5 encontra-se inactiva indicando deste modo que a expressão se encontra incorrecta. Assim quando a condição vectori & vectorj = vectorj se verifica a saída será "00000001", caso contrário "00000000".

Executar a operação vectori or vectorj

Para esta operação foi utilizado uma nova funcionalidade: o *Shifter* do bloco *Buffer/Shifter*. Esta irá permitir formar o resultado bit a bit. O resultado da operação é dado na saída Y5 a cada ciclo de relógio e *shiftado* de uma unidade, formando-se deste modo um vector com o resultado da operação or. O diagrama de fluxo de sinal correspondente encontra-se representado na figura seguinte:

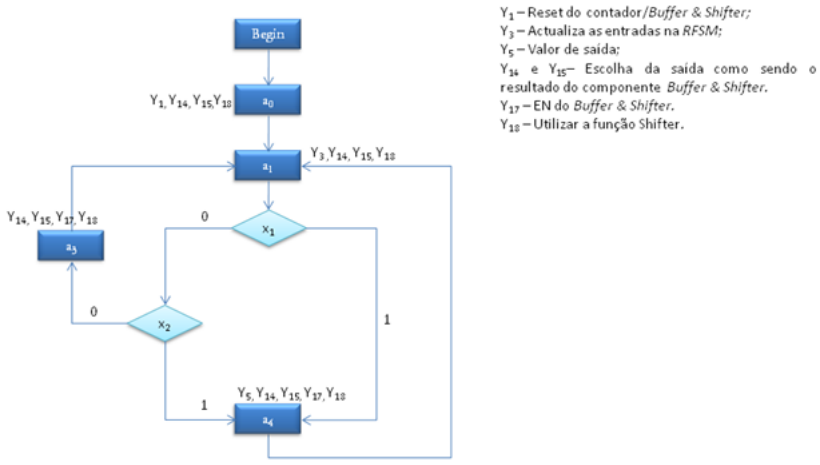

Figura A.13: Diagrama de fluxo de sinal para a instrução que permite executar a operação vectori or vectorj

Tal como a operação anterior, a sequência de operações inicia-se com uma operação de *reset*, no qual coloca o vector de saída a zero. De seguida são actualizadas as entradas da RFSM procedendo-se à análise destas. Começando pela primeira entrada, caso esta possua um valor lógico '1' imediatamente a maquina transita para o estado a4 uma vez que o resultado irá dar '1' sem que seja necessário analisar a segunda entrada. Caso

a primeira entrada possua o valor lógico '0' é necessário avaliar a segunda entrada. Se a segunda entrada possuir novamente o valor lógico '0' a máquina irá transitar para o estado a3 resultando uma saída Y5=0. A activação do sinal Y17 quer no estado a3, quer no estado a4 irá indicar ao bloco Buffer/Shifter que deverá receber o valor indicado por Y5 e executar um shift lógico. Caso a segunda entrada tome o valor '1' então o resultado da operação será '1' colocando assim a máquina no estado a4 sendo colocada neste a saída Y5 com o valor lógico '1'. Após o processamento de todas o valores obtêm-se o resultado na saída do bloco *Buffer/Shifter*.

Verificar se um vector apenas possui don't cares

Com esta operação pretende-se se saber se um dado vector possui apenas valores don't care. Para isso foi utilizado o sinal *Care1* que liga à entrada x3. Através deste é possível saber se a entrada x1 possui o valor *don't care* ou não. Foi utilizado o bloco *Counter/Comparator* para efectuar esta operação. O diagrama de fluxo de sinal encontra-se representado na figura seguinte:

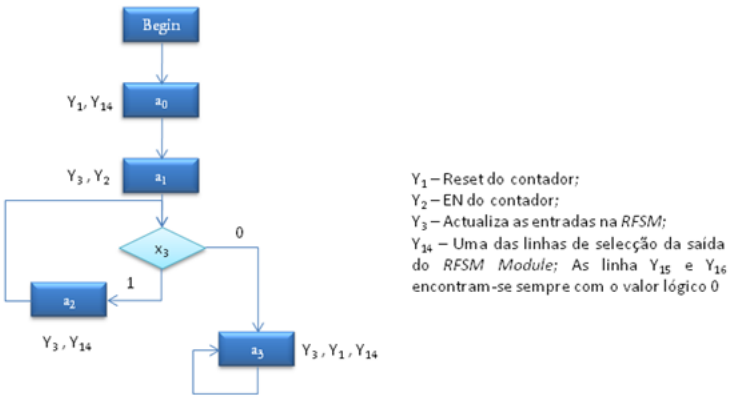

Figura A.14: Diagrama de fluxo de sinal para a instrução que permite verificar se um vector apenas possui don't cares

Inicialmente é colocado o contador com o valor 0. No estado seguinte é incrementado de uma unidade. Desta forma é considerado que o vector apenas possui *don't cares*. É então analisada a entrada x3 que indica se amostra actual possui o valor *don't care*. Se tal acontecer é analisada a amostra seguinte. Se todas as amostras possuírem *don't cares* a maquina encontrar-se-á sempre no estado a2 não sendo deste modo alterado o valor do contador que se encontra com o valor "00000001". Se uma das amostras possuir o valor lógico '0' a máquina irá transitar para o estado a3 fazendo um *reset* ao contador colocando na sua saída o valor "00000000". Esta irá manter-se no estado a3 mantendo-se neste até ao final da execução.

A.3 Instruções do processador de uso geral

O processador de uso geral utilizado após algumas modificações tem a seguinte estrutura. Na figura seguinte é já exibida a ligação ao processador desenvolvido.

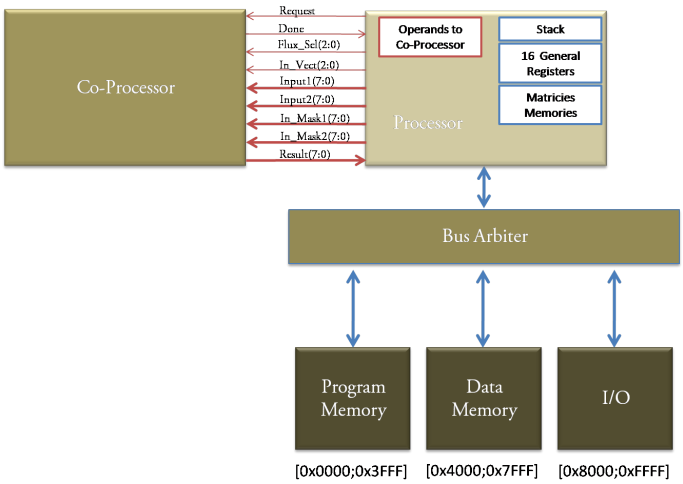

Figura A.15: Processador de uso geral e ligação ao co-processador

O *instruction set* do processador original foi igualmente alterado de forma a ser possível a realização dos algoritmos de satisfação booleana e cobertura de matrizes. Este encontra-se representado na tabela seguinte.

Instruções	Descrição
Aritméticas	
RADD, RD, R1, R2	Soma os registos R1 com R2 e guarda resultado em RD
RSUB, RD, R1, R2	Subtrai os registos R1 com R2 e guarda resultado em RD
RNOT, RD	Nega o registo RD
RAND, RD, R1, R2	Efectua a operação *and* entre os registos R1 e R2 e guarda resultado em RD
RROR, RD, R1, R2	Efectua a operação *or* entre os registos R1 e R2 e guarda resultado em RD
Load/Store	
LDR, RD, CONST	Carrega o valor CONST para o registo RD
LDA, RD, ADD_HIGH, ADD_LOW	Carrega para o registo RD o valor que se encontra na posição de memória ADD_HIGH+ADD_LOW
STA, RD, ADD_HIGH, ADD_LOW	Guarda o registo RD na posição de memória ADD_HIGH+ADD_LOW
Shift	
SLLi, RD	Efectua um shift lógico à esquerda do registo RD, sendo colocado um zero no bit menos significativo
SLRi, RD	Efectua um shift lógico à direita do registo RD, sendo colocado um zero no bit menos significativo
ROLi, RD	Efectua um shift rotativo à direita do registo RD, ou seja o bit mais significativo passa a bit menos significativo
RORi, RD	Efectua um shift rotativo à esquerda do registo RD, ou seja o bit menos significativo passa a bit mais significativo
Jump/Branch	
JMPi, ADD_HIGH, ADD_LOW	Efectua um salto incondicional para o endereço ADD_HIGH+ADD_LOW
BEQ, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja igual ao registo 13
BNE, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD não seja igual ao registo 13
BGE, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja maior ou igual do que o registo 13
BG, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja maior do que o registo 13
BLE, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja menor ou igual do que o registo 13
BL, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja menor do que o registo 13
BEZ, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD seja igual a zero
BNZ, RD, ADD_HIGH, ADD_LOW	Efectua um salto condicional para o endereço ADD_HIGH+ADD_LOW caso o registo RD não seja igual a zero
Matrix	
GetRow, RD, R1	Carrega a linha da matriz de dados indicada pelo registo R1 para o registo RD
GetRow, RD, R1	Carrega a coluna da matriz de dados indicada pelo registo R1 para o registo RD
GetCareRow, RD, R1	Carrega a linha da matriz de don't cares indicada pelo registo R1 para o registo RD
GetCareColumn, RD, R1	Carrega a coluna da matriz de don't cares indicada pelo registo R1 para o registo RD
DeleteRow, RD	Apaga a linha indicada pelo registo RD
DeleteColumn, RD	Apaga a coluna indicada pelo registo RD
GetRowMask, RD	Carrega para o registo RD a informação acerca das linhas apagadas da matriz
GetColumnMask, RD	Carrega para o registo RD a informação acerca das colunas apagadas da matriz
Stack	
IncStack	Guarda o valor actual dos registos e incrementa o valor do Stack_Pointer
DecStack	Decrementa o valor do Stack_Pointer de uma unidade e carrega os valores guardados para

Tabela A.1: *Instruction Set* do processador de uso geral

Apêndice B

Módulo CC1101

B.1 Sumário

Este apêndice apresenta algumas informações adicionais sobre o módulo wireless utilizado, bem como os desenhos das placas de circuito impresso desenvolvidas.

B.2 Restrições temporais

Interface SPI

Tal como foi visto é necessário ter atenção às restrições temporais na ligação com o módulo. Estas encontram-se apresentadas detalhadamente na figuraB.1 e tabelaB.1.

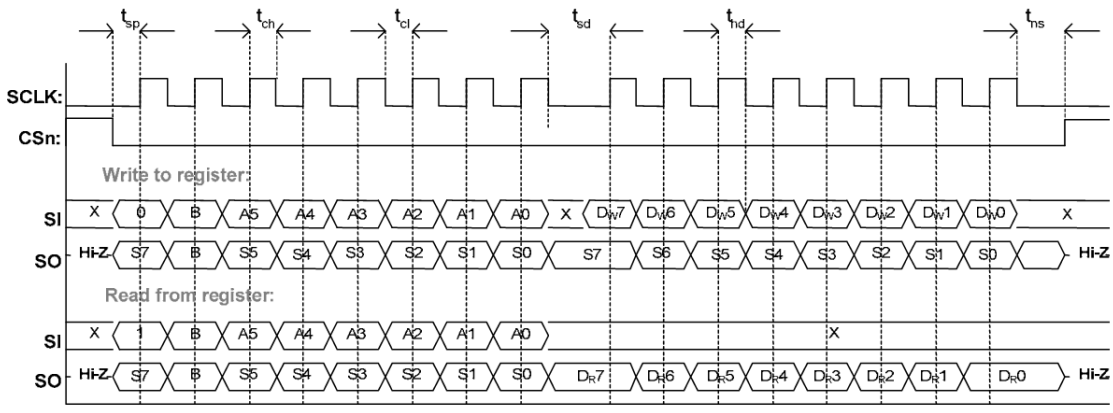

Figura B.1: Diagrama temporal do módulo CC1101 [24]

Parameter	Description		Min	Max	Units
f_{SCLK}	SCLK frequency 100 ns delay inserted between address byte and data byte (single access), or between address and data, and between each data byte (burst access). SCLK frequency, single access No delay between address and data byte SCLK frequency, burst access No delay between address and data byte, or between data bytes		- - -	10 9 6.5	MHz
$t_{sp,pd}$	CSn low to positive edge on SCLK, in power-down mode		150	-	µs
t_{sp}	CSn low to positive edge on SCLK, in active mode		20	-	ns
t_{ch}	Clock high		50	-	ns
t_{cl}	Clock low		50	-	ns
t_{rise}	Clock rise time		-	5	ns
t_{fall}	Clock fall time		-	5	ns
t_{sd}	Setup data (negative SCLK edge) to positive edge on SCLK (t_{sd} applies between address and data bytes, and between data bytes)	Single access Burst access	55 76	- -	ns
t_{hd}	Hold data after positive edge on SCLK		20	-	ns
t_{ns}	Negative edge on SCLK to CSn high.		20	-	ns

Tabela B.1: Restrições temporais do módulo CC1101 [24]

Comutação entre estados

A tabela seguinte mostra os tempos de atraso na transição entre estados de operação.

Description	XOSC Periods	26 MHz Crystal
IDLE to RX, no calibration	2298	88.4µs
IDLE to RX, with calibration	~21037	809µs
IDLE to TX/FSTXON, no calibration	2298	88.4µs
IDLE to TX/FSTXON, with calibration	~21037	809µs
TX to RX switch	560	21.5µs
RX to TX switch	250	9.6µs
RX or TX to IDLE, no calibration	2	0.1µs
RX or TX to IDLE, with calibration	~18739	721µs
Manual calibration	~18739	721µs

Tabela B.2: Tempos de atraso na comutação entre estados [24]

B.3 Mapeamento do módulo CC1101

A tabela seguinte mostra o respectivo mapeamento do módulo CC1101.

	Write		Read		
	Single Byte	Burst	Single Byte	Burst	
	+0x00	+0x40	+0x80	+0xC0	
0x00	IOCFG2				R/W configuration registers, burst access possible
0x01	IOCFG1				
0x02	IOCFG0				
0x03	FIFOTHR				
0x04	SYNC1				
0x05	SYNC0				
0x06	PKTLEN				
0x07	PKTCTRL1				
0x08	PKTCTRL0				
0x09	ADDR				
0x0A	CHANNR				
0x0B	FSCTRL1				
0x0C	FSCTRL0				
0x0D	FREQ2				
0x0E	FREQ1				
0x0F	FREQ0				
0x10	MDMCFG4				
0x11	MDMCFG3				
0x12	MDMCFG2				
0x13	MDMCFG1				
0x14	MDMCFG0				
0x15	DEVIATN				
0x16	MCSM2				
0x17	MCSM1				
0x18	MCSM0				
0x19	FOCCFG				
0x1A	BSCFG				
0x1B	AGCCTRL2				
0x1C	AGCCTRL1				
0x1D	AGCCTRL0				
0x1E	WOREVT1				
0x1F	WOREVT0				
0x20	WORCTRL				
0x21	FREND1				
0x22	FREND0				
0x23	FSCAL3				
0x24	FSCAL2				
0x25	FSCAL1				
0x26	FSCAL0				
0x27	RCCTRL1				
0x28	RCCTRL0				
0x29	FSTEST				
0x2A	PTEST				
0x2B	AGCTEST				
0x2C	TEST2				
0x2D	TEST1				
0x2E	TEST0				
0x2F					
0x30	SRES		SRES	PARTNUM	Command Strobes, Status registers (read only) and multi byte registers
0x31	SFSTXON		SFSTXON	VERSION	
0x32	SXOFF		SXOFF	FREQEST	
0x33	SCAL		SCAL	LQI	
0x34	SRX		SRX	RSSI	
0x35	STX		STX	MARCSTATE	
0x36	SIDLE		SIDLE	WORTIME1	
0x37				WORTIME0	
0x38	SWOR		SWOR	PKTSTATUS	
0x39	SPWD		SPWD	VCO_VC_DAC	
0x3A	SFRX		SFRX	TXBYTES	
0x3B	SFTX		SFTX	RXBYTES	
0x3C	SWORRST		SWORRST	RCCTRL1_STATUS	
0x3D	SNOP		SNOP	RCCTRL0_STATUS	
0x3E	PATABLE	PATABLE	PATABLE	PATABLE	
0x3F	TX FIFO	TX FIFO	RX FIFO	RX FIFO	

Tabela B.3: Mapeamento dos diversos registos do módulo CC1101

B.4 Command Strobes

A tabela seguinte ilustra os diversos command strobes possíveis no módulo CC1101 bem como a descrição da sua funcionalidade.

Address	Strobe Name	Description
0x30	SRES	Reset chip.
0x31	SFSTXON	Enable and calibrate frequency synthesizer (if MCSM0.FS_AUTOCAL=1). If in RX (with CCA): Go to a wait state where only the synthesizer is running (for quick RX / TX turnaround).
0x32	SXOFF	Turn off crystal oscillator.
0x33	SCAL	Calibrate frequency synthesizer and turn it off. SCAL can be strobed from IDLE mode without setting manual calibration mode (MCSM0.FS_AUTOCAL=0)
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and MCSM0.FS_AUTOCAL=1.
0x35	STX	In IDLE state: Enable TX. Perform calibration first if MCSM0.FS_AUTOCAL=1. If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x38	SWOR	Start automatic RX polling sequence (Wake-on-Radio) as described in Section 19.5 if WORCTRL.RC_PD=0.
0x39	SPWD	Enter power down mode when CSn goes high.
0x3A	SFRX	Flush the RX FIFO buffer. Only issue SFRX in IDLE or RXFIFO_OVERFLOW states.
0x3B	SFTX	Flush the TX FIFO buffer. Only issue SFTX in IDLE or TXFIFO_UNDERFLOW states.
0x3C	SWORRST	Reset real time clock to Event1 value.
0x3D	SNOP	No operation. May be used to get access to the chip status byte.

Tabela B.4: Endereço dos *command strobes* e sua função [24]

B.5 Esquema do Kit CC1101EMK433

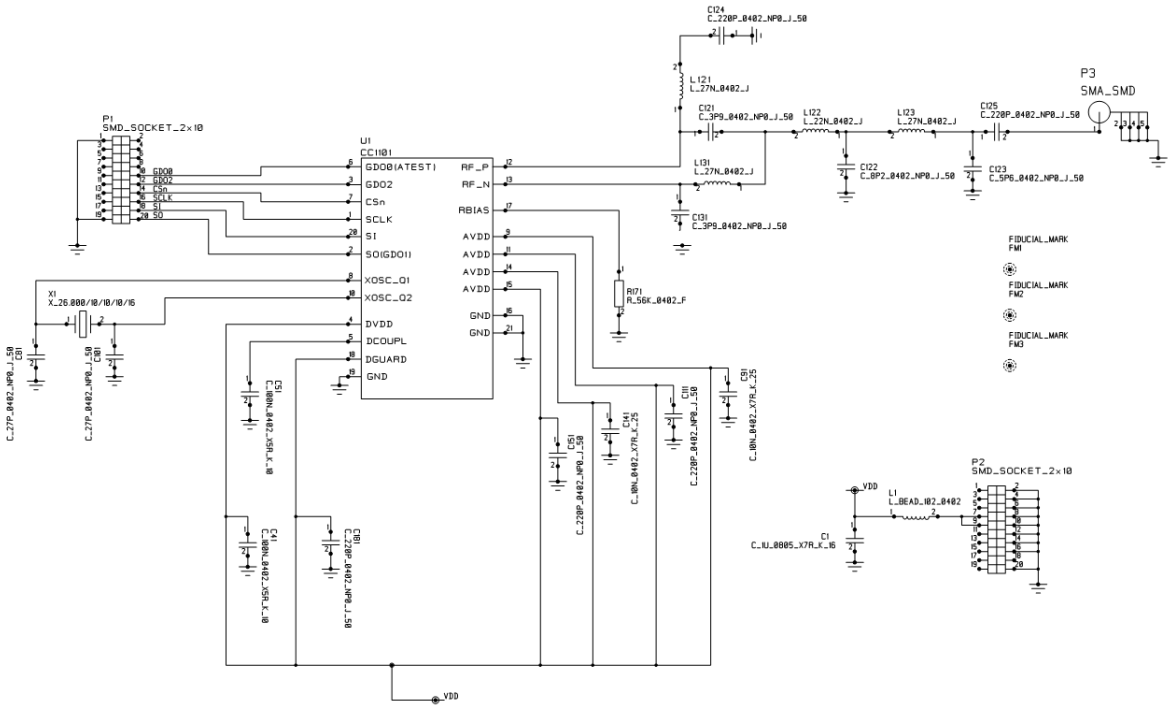

Figura B.2: Esquema do Kit CC1101EMK433 adaptado para uma frequência de trabalho de 433MHZ

B.6 Placas de circuito impresso construídas

Placa de *Debug*

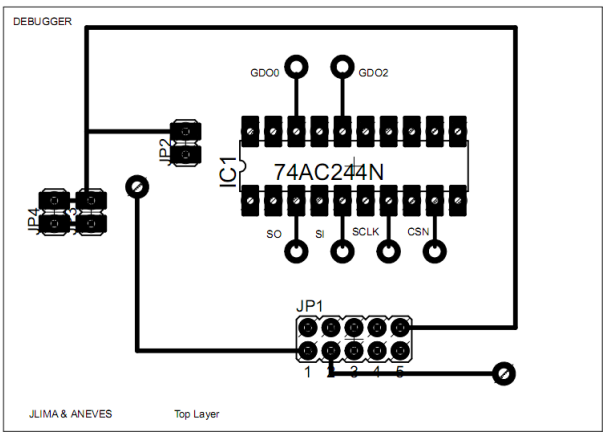

Figura B.3: Placa de *Debug* - *Top Layer*

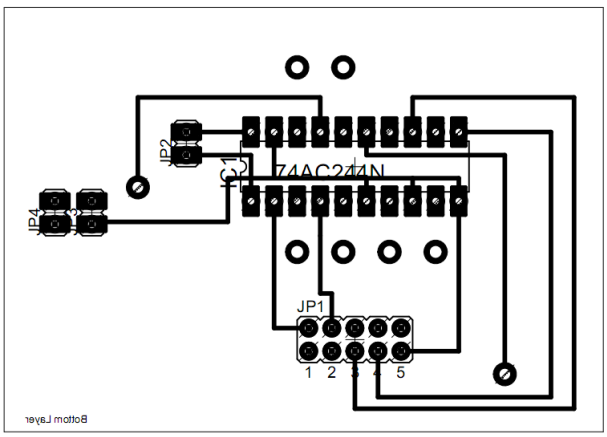

Figura B.4: Placa de *Debug* - *Bottom Layer*

Placa de ligação do módulo wireless às placas de desenvolvimento

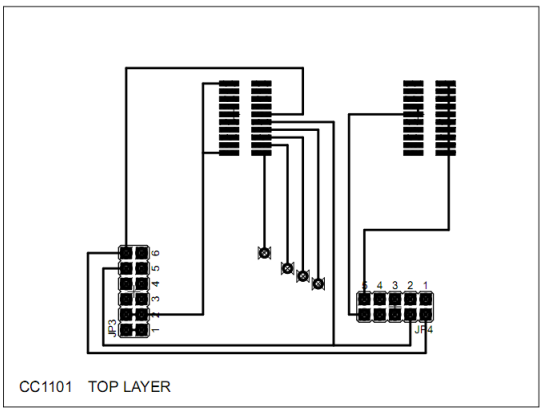

Figura B.5: Placa de ligação do módulo wireless às placas de desenvolvimento - *Top Layer*

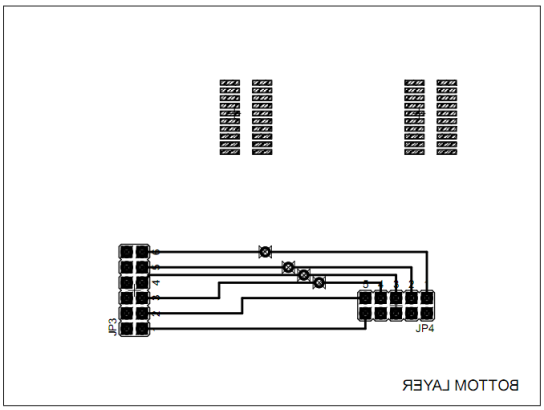

Figura B.6: Placa de ligação do módulo wireless às placas de desenvolvimento - *Bottom Layer*

Bibliografia

[1] http://www.willsmith.org/collectible_chips/.

[2] http://www.beis.de/Elektronik/DPLCM/DPLCM.html.

[3] http://www.goldmine-elec products.com/prodinfo.asp?number=A20321.

[4] Xilinx. Programmable logic design - quick start handbook, Junho 2006.

[5] Xilinx. Revolutionary architecture for the next generation platform fpgas, Dezembro 2003.

[6] http://www.xilinx.com/company/gettingstarted/index.htm.

[7] http://labspace.open.ac.uk/mod/resource/view.php?id=360467.

[8] http://www.designreuse.com/news/16992/ttpcontrolleripalteracostcyclonefpgafamiliesaerospace applications.html.

[9] http://www.semiconductors.bosch.de/en/20/ttcan/index.asp.

[10] http://www.digilentinc.com/Products/Detail.cfm?NavTop=2&NavSub=521&Prod=NETFPGA.

[11] http://www.phxmicro.com/Training/AMCC/PPC440Core.html.

[12] http://www.hardwarezone.com.au/reviews/view.php?cid=10&id=1725.

[13] http://www.networkworld.com/community/node/20520.

[14] Petterson A. David; Hennessy L. John. *Computer Organization and Design, The hardware and software interface*. 3th edition, 2005.

[15] http://www.xilinx.com/itp/xilinx10/isehelp/ise_c_project_navigator_overview.htm.

[16] Xilinx. *Xilinx ISE Design Suite 10.1 Software Manuals*, 2008.

[17] http://www.cadinformatique.com/images/ModelSim_Des_big.jpg.

[18] http://www.martijnthe.nl/publications/electronic-toy prototype/index.html.

[19] Celoxica. *RC 10 Manual*. www.celoxica.com.

[20] http://www.arl.wustl.edu/ lockwood/class/cse566 f06/projects/photos.html.

[21] Digilent. *Nexys 2 Manual*. www.digilentinc.com.

[22] 451&Prod=NEXYS2 http://www.digilentinc.com/Products/Detail.cfm?NavPath=2, 400.

[23] Valery Sklyarov. Reconfigurable models of finite state machines and their implementation in fpgas. *Journal of Systems Architecture*, 47:1043–1064, 2002.

[24] Texas Instruments. *CC1101 Low-Cost Low-Power Sub-1GHz RF Transceiver Datasheet*. www.ti.com.

[25] Mouser Electronics.

[26] http://www.embedded.com/columns/technicalinsights/184400800?_requestid=190366.

[27] CC1101 Design Note DN503.

[28] http://www.xilinx.com/products/v6s6.htm.

[29] Iouliia Skliarova. *Arquitecturas reconfiguráveis para problemas de optimização combinatória*. PhD thesis, Universidade de Aveiro, 2004.

[30] ece.ut.ac.ir/classpages/S84/LogicLab/FPGA.pdf.

[31] Xilinx. *Xilinx CPLD Applications Handbook*, 2005.

[32] http://www.xilinx.com/company/history.htm.

[33] Xilinx. *EDK Concepts, To and Techniques A Hands-On Guide to Effective Embedded System Design*, 2009.

[34] http://www.xilinx.com/company/gettingstarted/index.htm.

[35] http://www.xilinx.com/esp/index.htm.

[36] http://bwrc.eecs.berkeley.edu/People/Grad_Students/rcshah/documents/ttcan.ppt.

[37] http://www.interfacebus.com/Programmable_Logic.html.

[38] http://www.xilinx.com/products/virtex5/.

[39] http://www.xilinx.com/products/spartan3a/.

[40] A. Dewey and A. Gadient. Vhdl motivation. *IEEE Design & Test of Computers*, 3(2):12–16, April 1986.

[41] Narinder Lall Eric Cigan. Integrating matlab algorithms into fpga designs into fpga designs, 2005.

[42] http://encyclopedia2.thefreedictionary.com/instruction+set+architecture.

[43] http://wiki.answers.com/Q/Difference_between_von_newman_and_harvard_computer_arch

[44] http://encyclopedia2.thefreedictionary.com/addressing+mode.

[45] http://www.bitpipe.com/tlist/Coprocessors.html.

[46] http://www.embedded.com/columns/technicalinsights/184400800?_requestid=190366.

[47] Xilinx. *PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II, and Virtex-II Pro FPGAs*, 2008.

[48] http://www.xilinx.com/products/design_resources/proc_central/microblaze_arc.htm.

[49] Xilinx. *Embedded Processor Block in Virtex-5 FPGAs Reference Guide*, 2009.

[50] http://www.eda experten.de/pdf/ModelSimDesignerDS60.pdf.

[51] http://www.cadinformatique.com/HDL_la_simulation.htm.

[52] Peter J. Ashenden. *Digital Design, An Embedded Systems Approach Using VHDL.* 2008.

[53] http://www-md.e-technik.uni rostock.de/lehre/vlsi_i/proc8/index.html.

[54] Skliarova I. ; Sklyarov V. Design methods for fpga-based implementation of combinatorial search algorithms.

[55] http://sweet.ua.pt/ a16360/index.html.

[56] http://www.terabeam.com/support/calculations/fresnel zone.php.

Nomenclature

ASIC - Application Specific Integrated Circuits

CAD - Computer-Aided Design

CAM - Content-addressable memory

CCA - Clear Channel Assessment

CI - Circuito Integrado

CLB - Configurable Logic Blocks

CPHA - Clock Phase

CPLD - Complex Programmable Logic Device

CPLD - Complex Programmable Logic Device

CPOL - Clock Polarity

CPU - Central Processing Unit

CS - Chip Select

DCM - Digital Clock Manager

DRAM - Dynamic Random Access Memory

EDK - Embedded Development Kit

EEPROM - Electrically Erasable Programmable Read Only

EPROM - Erasable Programmable Read Only Memory

FIFO - First In, First Out

FPGA - Field Programmable Gate Array

FSM - Finite State Machine

GAL - Generic Logic Array

HDL - Hardware Description Languages

HSPDA - High-Speed Downlink Packet Access

IOB - Input/Output Block

IP - Intelectual Property

ISA - Instruction Set Architecture

ISE - Integrated Synthesis Environment

ISE - Integrated Synthesis Environment

LNA - Low Noise Amplifier

LSI - Large Scale Integration

LSI - Large Scale Integration

LVDS - Low-voltage differential signaling

Matlab - MATrix LABoratory

MISO - Master Input, Slave Output

MOSI - Master Output, Slave Input

MPEG - Moving Picture Experts Group

MSI - Medium Scale Integration

MSI - Medium Scale Integration

NoC - Network On Chip

PA - Power Amplifier

PAL - Programmable Array Logic

PCI - Peripheral Component Interconnect

PDA - Personal Digital Assistants

PLA - Programmable Logic Array

PLD - Programmable Logic Device

PROM - Programmable ROM

RAM - Random Access Memory

RF - Rádio Frequência

RFSM - Reconfigurable Finite State Machine

RISP - Reduced Instruction Set Processor

ROM - Read Only Memory

SCLK - SPI Clock

SoC - System On Chip

SPI - Serial Peripheral Interface

SPLD - Simple Programmable Logic Device

SPLD - Simple Programmable Logic Device

SRAM - Static Random Access Memory

SS - Slave Selected

SSI - Small Scale Integration

SSI - Small Scale Integration

TTCAN - Time Triggered Controller Area Network

TTCAN - Time-Triggered CAN

TTP - Time-Triggered Protocol

UCF - User Constraint File

VISC - Variable Instruction Set Computer

VLSI - Very Large Scale Integration

VLSI - Very Large Scale Integration

WCDMA - Wideband Code Division Multiple Access

WiMAX - Worldwide Interoperability for Microwave Access